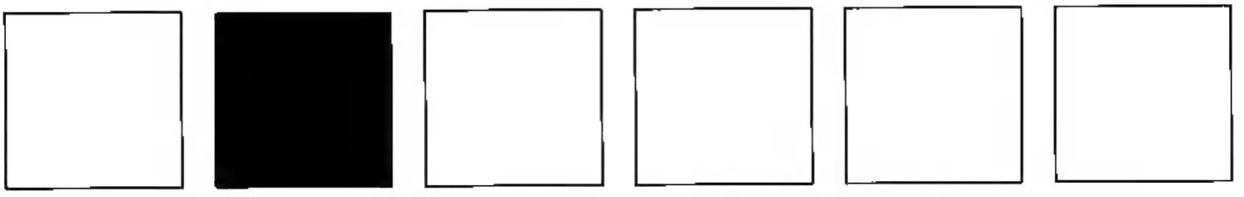
HEWLETT PACKARD

HP 82490A

HP-41 Translator Pac

Owner's Manual

For the HP-71



HEWLETT PACKARD

HP 82490A
HP-41 Translator Pac

Owner's Manual

For the HP-71

January 1985

Developed and written for Hewlett-Packard by
Dr. William C. Wickes

82490-90001

Printed in U.S.A.

Introducing the HP-41 Translator Pac

The HP 82490A HP-41 Translator Pac consists of:

- The HP-41 Translator Pac module, which contains a system of programs for the HP-71 designed to allow you to convert programs written for the HP-41 for use on the HP-71. You can then run the "translated" programs on your HP-71, taking advantage of the HP-71's enhanced speed and accuracy.
- A keyboard overlay, designed to customize your HP-71 keyboard for use with the HP-41 Translator Pac.

With the HP-41 module installed in your HP-71, you can:

- Use the HP-71 as an HP-41 calculator for keyboard operations and program execution. All HP-41 calculator features, including the alpha register, are available from the HP-71 keyboard.
- Write programs on the HP-71 in HP-41 user-programming language.
- Transfer programs automatically from the HP-41 to the HP-71 via HP-IL. (This requires HP-IL modules for both calculators.)
- Extend the HP-41 user language, either by adding HP-41 extension module functions not included in the HP-41 Translator Pac, or by adding new functions of your own design.

In short, the HP-41 Translator Pac allows you to retain the calculator and programming features of the HP-41, and to use your HP-41 programs, as you move into the more sophisticated and powerful world of the HP-71.

The HP-41 function set included in the HP-41 Translator Pac includes:

- All functions built into the HP-41C and HP-41CV calculators.
- Additional numeric, flag, and alpha data manipulation functions from the HP-41CX and the HP 82180A Extended Functions/Memory Module.
- All of the HP-IL printer functions from the HP 82160A HP-IL Module for the HP-41 except the special graphics functions.

Not included in the HP-41 Translator Pac are the time (TIME is included), date, stopwatch, alarm, and extended memory functions from the HP-41CX, or any other functions from HP-41 extension modules.

Any HP-41 program that uses functions included in the pac can be translated and run on the HP-71. HP-41 programs and keyboard operations that are executed on the HP-71 yield the same results as those obtained with the HP-41, except that:

- The HP-71 provides two additional mantissa digits and and one more exponent digit than the HP-41. This affects both the accuracy of numeric results and the allowed range of numbers.
- Unlike the HP-41, The HP-41 Translator Pac does not provide the ability to choose the digit separator and radix format.
- There are some differences in keyboard entry methods that arise from the different keyboards of the two computers and from the somewhat different styles of RPN arithmetic associated with HP-71 FORTH and the HP-41.

Appendix E contains a list of the HP-41 functions provided by the HP-41 Translator Pac, and description of the differences between the HP-41 and the HP-41 Translator Pac.

The HP-41 Translator Pac is based on a FORTH language system built into the pac. FORTH is a computer language that shares many characteristics with the HP-41 user-programming language, but is more widely recognized and applied to a greater variety of computers. You do not need to learn FORTH to use the HP-41 translator, but programmers familiar with FORTH can use the system to write HP-71 applications entirely in FORTH.

Contents

How to Use This Manual

This manual contains information you need in order to use the HP-41 Translator Pac. You will need to read portions of this manual if:

- You are familiar with the HP-71, but have little or no programming experience on the HP-41. For example, you may want to use the HP-41 Translator Pac to translate and run HP-41 programs purchased from the Users' Library.
- You want to use your HP-71 as an RPN calculator.
- You are familiar with both the HP-41 and the HP-71, and have written or acquired HP-41 programs you wish to translate and run on the HP-71.

If you will be using the HP-41 Translator Pac to run purchased HP-41 programs, you do not need to read this manual cover-to-cover. If you are not familiar with operation of the HP-41, you should read appendix F before continuing with your reading. Appendix F, plus the instructions accompanying your purchased programs, may contain enough information to run those programs. At times, you may need to refer to portions of this manual for additional information about differences between the HP-71 and the HP-41, but you will not need a complete understanding of how the translator works.

Sections 1 through 5, and appendixes A through E assume that you are familiar with the general operation of both the HP-71 and HP-41. The use of an HP-41 is not required, but since the HP-41 Translator Pac emulates HP-41 operation on the HP-71, we assume that you are accustomed to using the HP-41. If you will be translating your own HP-41 programs, or if you intend to use your HP-71 as an RPN calculator, you will need to carefully read the appropriate sections:

- Sections 1 and 2 describe the HP-41 features provided by the pac. Section 1 shows you how to install the HP-41 module and enter the HP-41 "environment," where the HP-71 behaves like (emulates) an RPN calculator. It then guides you through several examples using the HP-41 "emulator." Section 2 covers the HP-41 emulator in greater detail.
- Section 3 descibes how to use the pac to transfer HP-41 programs written on the HP-41 to the HP-71. In addition, section 3, describes how to run HP-41 user-language programs written using the HP-71 Text Editor.
- Section 4 describes how to use the HP-71 Text Editor to write or edit HP-41 user-language programs for running on the HP-71.
- Section 5 is intended for users familiar with the FORTH language. It describes the FORTH system that underlies the HP-41 emulator. This is a complete FORTH system; with it, you can extend the emulator function set, write HP-71 applications entirely in FORTH, or write programs that mix routines written in BASIC and FORTH. Section 5 does not provide a FORTH tutorial, but does list some standard references for programmers who wish to learn about the language.

There are also several appendixes for your reference:

- Appendix A, "Owner's Information," includes warranty and service information.
- Appendix B contains descriptions of error messages.
- Appendix C describes the BASIC keywords added to the HP-71 BASIC language by the pac.
- Appendix D contains an alphabetical list of the FORTH words included in the built-in FORTH dictionary.
- Appendix E "Summary of the HP-41 Emulator Features," describes the HP-41 functions and capabilities that are included in the emulator, and lists the general and specific differences between emulator functions and their HP-41 equivalents.
- If you are not familiar with the HP-41, Appendix F will get you started in translating and running HP-41 programs.

If you intend to use the HP-41 Translator Pac to transfer programs from the HP-41 to the HP-71 via HP-IL, you must know how to install the HP-IL modules into the two computers. If your system includes peripheral devices such as printers and mass storage devices, you will need to know how to connect them. If necessary, refer to the owner's manuals for the interfaces and devices for further instructions.

Section 1

Getting Started

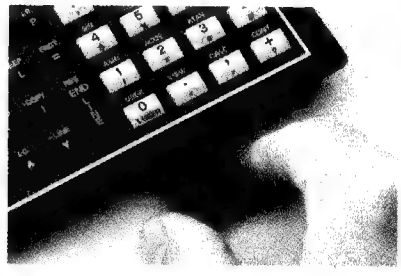
Installing and Removing the HP-41 Translator Pac Module

The HP-41 Translator Pac module can be plugged into any of the four ports on the front edge of the HP-71.

CAUTIONS

- Be sure to turn off the HP-71 (press f ON) before installing or removing a module.
- Whenever you remove a module to make a port available for another module, be sure to turn the HP-71 on and then off *while the port is empty* before installing the new module.
- Do not place fingers, tools, or other foreign objects into any of the ports. Such actions could result in minor electrical shock hazard and interference with pacemaker devices worn by some persons. Damage to port contacts and internal circuitry could also result.
- Never install the HP 82490A HP-41 Translator Pac module and the HP 82441A FORTH/Assembler ROM module into the HP-71 at the same time. The presence of both modules can cause the system to operate incorrectly.

To insert the module, hold the HP-71 with the keyboard facing up and the module with the label facing up. Push the module into the port until it snaps into place. Be sure to observe the precautions described above.



To remove the module, use your fingernails to grasp the module by the lip on the bottom of its front edge. Pull the module straight out of the port. Install a blank module in the port to protect its contacts.

What the HP-41 Translator Pac Does

The HP-41 capabilities provided by the HP-41 Translator Pac fall into two categories:

- The HP-41 **emulator** enables the HP-71 to recognize all HP-41 keyboard operations and program execution commands, and provides the HP-41 RPN arithmetic stack, data and alpha registers, and user flags. The HP-71 key file KEYS41 makes the HP-71 respond to keystrokes in a key-per-function manner very similar to the actual HP-41.
- The HP-41 **translator** consists of two BASIC programs, READ41 and TRANS41. READ41 transfers HP-41 programs in HP-41 memory to text files in HP-71 memory via HP-IL. TRANS41 translates HP-71 text files containing HP-41 user language programs (created either by READ41 or with the text editor) into a form suitable for loading into the HP-41 emulator program memory. TRANS41 also loads translated programs into emulator memory, where they are ready to run.

Starting the Emulator

Follow this procedure to start the emulator. Steps #2 and #3 are optional; they are included here so you can follow along with the examples in this section.

1. Turn on the HP-71 with the HP-41 Translator Pac module installed in any port.
2. Type FIX 4 [END LINE] to set the display format to four decimal places to the right of the decimal point.
3. If USER mode is active, press [f][USER] to disable USER mode for now.
4. Type HP41 [END LINE]. The system displays:

   HP-41 EMULATOR 1A

   to indicate that you have entered the HP-41 environment.
5. If this is the first time you have activated the emulator, the system displays the prompt:

   SIZE(max, nnn)?

   asking you to input the number of HP-41 data registers you wish to create. This can be any number in the range 1 through *nnn*. For now, type:

   100 [END LINE]

   The HP-71 displays the contents of the HP-41 emulator X-register, which should be 0.0000 if you are using the emulator for the first time.

A First Try

Now you're ready to try some HP-41 keyboard operations. You will see that the emulator approximates the HP-41 closely, but there are differences arising from the different keyboards and capabilities of the two calculators.

Starting with something simple, let's try adding 1 plus 2, and multiplying the result by 3. On the HP-41, you would press:

[1][ENTER↑][2][+][3][*]

to see the result 9.0000. Moreover, you would see intermediate results in the X-register after pressing [ENTER↑] and [+].

There is no [ENTER↑] key on the HP-71. In fact, the HP-41 keyboard and the HP-71 keyboard are so fundamentally different that the HP-41 Translator Pac emulates only the "spirit" of the HP-41 arithmetic style, and does not attempt to mimic exact key-by-key sequences. To see what this means, type:

[1][SPC][2][SPC][+][SPC][3][SPC][*][END LINE]

on one line. Pressing [END LINE] displays the result 9.0000, just as on the HP-41. Notice that the order of operations is identical in both the HP-41 and the HP-71 key sequences, but the actual keystrokes differ. The idea here is that you will perform HP-41 RPN arithmetic on the HP-71 in a manner that takes advantage of the capabilities of the HP-71, particularly its ability to remember and execute a sequence of commands all together. Emulation of the HP-41 stack is discussed in much more detail in Section 2. The HP-71 keystroke sequence above is less efficient than its HP-41 counterpart because you must type spaces to separate each command or number entered, plus the final [END LINE]. However, you can use key assignments to make the HP-41/HP-71 correspondence even closer. Type KEYS41 [END LINE]. This causes the file KEYS41 included in the HP-41 Translator Pac to be merged into the HP-71 keys file. Press f-[USER] to activate USER mode.

Now try the arithmetic example again:

HP-71 Emulator		HP-41	
Keystrokes	**Display**	**Keystrokes**	**Display**
[1]	1	[1]	1_
[END LINE]	1.0000	[ENTER↑]	1.0000
[2]	2	[2]	2_
[+]	3.0000	[+]	3.0000
[3]	3	[3]	3_
[*]	9.0000	[*]	9.0000

In this example, the HP-71 [END LINE] key plays the role of the HP-41 [ENTER↑] key; otherwise the operations and results are the same on both calculators.

Another HP-41 key not present on the HP-71 keyboard is the [ALPHA] key. There are three ways to activate ALPHA mode:

- Type AON [END LINE]
- Type $ ([9][4]) [END LINE]
- When the KEYS41 file is the current keys file and USER mode is active, press [f][END LINE].

The HP-71 AC annuciator is displayed to indicate that the alpha keyboard is now active. The rest of the display is blank, because the alpha register is empty. Press the [H] key:

H_

to enter the letter H into the alpha register. The _ prompt indicates that subsequent keystrokes will be appended to the alpha string just as on the HP-41. Press [E][L][L][O][X]:

HELLOX_

Too many characters? Press the [←] key:

HELLO_

Now, press [↓] to display:

ASTO

and press [X] [END LINE] to execute the ASTO X function. Executing ASTO X stores the alpha data HELLO into the X-register. Note that the append prompt has disappeared. If you press another character key now, HELLO disappears, and the new character is displayed followed by an append prompt. To restore the append mode without deleting HELLO, press the [→] key, which plays the role of the HP-41 append key.

To return to execute mode, press [END LINE]. The AC annunciator turns off, and HELLO remains in the display (the contents of the X-register).

Try a few more familiar HP-41 operations. (You might wish to turn off USER mode to avoid activating assigned keys not yet discussed.) In general, executing a function involves typing out the HP-41 function name and parameters you wish to try and pressing [END LINE]. The HP-71 alphanumeric keyboard allows you to type the function names without using ALPHA mode. For example, compute the sine of 90°:

Keystrokes	Display	
DEG [END LINE]	no change	Sets degrees mode.
90	90▒	Digit entry.
[END LINE]	90.0000	Digit entry terminated.
SIN	SIN▒	Type function name.
[END LINE]	1.0000	Sine of 90°.

When the function specifies an HP-41 register, type the function name, a space, and the register number (or letter, for stack registers), terminating with [END LINE]. For example, to store the number 27.6 into register 5:

Keystrokes	Display	
27.6	27.6▒	Digit entry not terminated.
[END LINE]	27.6000	Digit entry terminated.
STO 5 [END LINE]	27.6000	Value 27.6 is stored in register 5.
CLX [END LINE]	0.0000	Clears the X-register.
RCL 5 [END LINE]	27.6000	Recalls contents of register 5.

A similar rule applies for functions such as SF and FIX.

Keystrokes	Display	
SF 23 [END LINE]	27.6000	Sets flag 23. The previous contents of the X-register are displayed.
FS? 23 [END LINE]	YES	Flag 23 is set.
FIX 7 [END LINE]	27.6000000	Display is formatted for seven decimal places.

You can type any number of functions together (always separating each command with one or more spaces) before hitting [END LINE], up to a maximum of 96 characters in each command string. The HP-71 command stack is fully functional in the HP-41 system, so that you can recall, edit, and re-execute previous commands.

Keystrokes	Display	
FIX 6 [END LINE]	27.600000	Display is formatted for six decimal places.
45 [END LINE]	45.000000	Digit entry.
SIN FIX 4 [END LINE]	.7071	Enter command string to see the sine of 45° to four decimal places.
[↑]	\45	Displays previously executed command.
[↑]	\FIX 6	Displays previously executed command.
[END LINE]	.707107	Executes FIX 6.
FIX 4 [END LINE]	.7071	Returns display format to FIX 4.

The power of the underlying HP-71 and the FORTH language system allow the pac to introduce additional functions that have no HP-41 counterpart. For example, type:

```
1 2 3 4 + 5 STD STACK [END LINE].
```

You will see all five registers of the RPN arithmetic stack at once:

```
1  2  7  5  |  4
↑  ↑  ↑  ↑     ↑
T  Z  Y  X     Last X
```

STD is a new HP-41 emulator function that implements the HP-71 *standard display* mode. STACK is a new function that instructs the HP-71 to display the entire RPN stack after each command line, instead of just the X-register. The stack is displayed left-to-right, registers T → X. The LAST X register is separated from the others by the | character.

To restore the normal X-register display, type XONLY [END LINE]—another new function. If you wish to exit standard display mode, execute another display function, for example, FIX 4.

Continuing Your Reading

This brief introduction presented an overview of the HP-41 emulator contained in the HP-41 Translator Pac. The rest of this manual presents a more thorough coverage of the pac's features:

- To learn more about how the the HP-41 emulator implements HP-41 functions, registers, and keystrokes, read Section 2.
- To learn how to create files executable by the HP-41 emulator from HP-41 user-language programs written using the HP-41 or the HP-71 editor, read Section 3.
- To learn how to use the HP-71 editor, read Section 4.
- If you intend to program the HP-71 in FORTH, read Section 5.

Appendixes A through E contain additional reference information. Appendix F summarizes the procedures to follow for using HP-41 programs on the HP-71.

Section 2

HP-41 Operation in Detail

Introduction

This section describes how to use the HP-41 emulator to run HP-41 programs and perform RPN calculations on your HP-71. The great majority of the HP-41 functions provided by the emulator work the same way as their HP-41 counterparts, and need no specific description here. The principal difference between the HP-41 and the HP-41 emulator is in their user interfaces—that is, the steps you must use to execute functions.

The HP-41 uses a key-per-function, one-function-at-a-time input style. The HP-41 emulator matches the HP-41 in the logical sequence of operations, but takes advantage of the additional capabilities of the HP-71 to provide a more flexible, multi-function command line method of keyboard input. For experienced HP-41 users, the emulator may take a little getting used to, but you will quickly come to appreciate the advantages of the emulator style.

Environments

With the HP-41 Translator Pac installed in the HP-71, you have at your disposal what at first may seem a confusing variety of commands and modes. The HP-71 retains all of its normal capabilities—immediate execute BASIC, CALC mode, BASIC program execution. The pac adds not only the HP-41 emulator and translator programs, but also the FORTH system that supports the HP-41 functionality. As you will see, the FORTH system and the HP-41 emulator operate somewhat differently from HP-71 BASIC.

The term *environment* is used to describe the various states of the HP-71. When the HP-71 is operating in a particular environment, it interacts with the user and responds to commands in a special way characteristic of that environment. Commands and operations from other environments are available only indirectly or not at all. The environments available with the HP-41 Translator Pac module installed are:

- **The BASIC environment.** This is the standard HP-71 environment, in which you can specify immediate-execute BASIC commands, run programs, manipulate files, make key assignments, etc. The prompt > indicates that the HP-71 is ready for input.
- **HP-71 CALC mode.** The HP-41 Translator Pac does not affect the use of CALC mode.

- **The HP-41 environment.** This is a new environment provided by the HP-41 Translator Pac. When you operate the HP-71 from the HP-41 environment, the calculator mimics the display behavior of an HP-41. That is, you normally see the contents of the X-register after every command or string of commands. You can also turn on ALPHA mode to display the current contents of the alpha register. In ALPHA mode, the full range of HP-41 alpha keys and commands are available. As on the HP-41, error messages and the YES or NO keyboard response to test functions can temporarily replace the X- or alpha register display.

  In the HP-41 environment, most of the pac's built-in FORTH words are available, but BASIC commands can only be executed indirectly. We will use the term *emulator memory* to refer to the portion of HP-71 memory used within the HP-41 environment to contain HP-41 programs and data.

- **The FORTH environment.** This environment is also added to the HP-71 by the HP-41 Translator Pac. You will not need to know FORTH to operate the HP-41 emulator, but advanced programmers can use the FORTH environment to extend the HP-41 emulator capabilities and to program HP-71 applications in FORTH. In the FORTH environment, the message `OK < 0 >` replaces the BASIC prompt `>`. In the FORTH environment, you can type FORTH commands and program the HP-71 in the FORTH language; the HP-71 does not recognize BASIC commands or most HP-41 functions.

The relationship between the three environments is shown in figure 2-1:

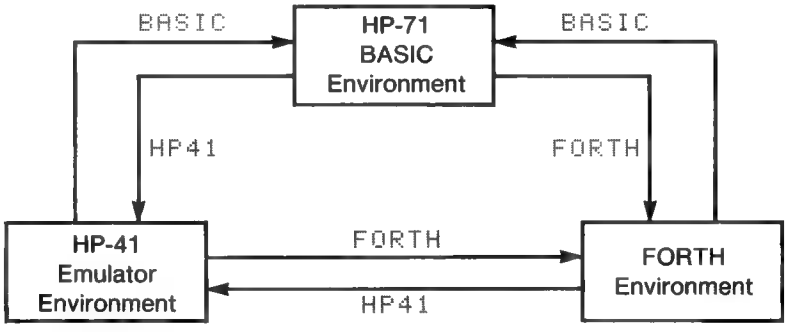


Figure 2-1. Keywords for Changing Environments

Each environment has two keywords that allow you to change environments, as shown by the words above the arrows in the diagram. To move from one environment to another, type the name of the new environment followed by [END LINE].

Entering and Exiting the HP-41 Environment

You can enter the HP-41 environment from either the BASIC or the FORTH environments by typing the keyword HP41. The HP-71 displays the message:

```
HP-41 EMULATOR 1A
```

indicating that you are ready to begin HP-41 operations. Whenever you enter the HP-41 environment, the HP-71 automatically carries out the following startup procedures:

- The display mode (FIX, SCI, ENG, or STD), angular mode (DEG or RAD), and number of display digits are set to match the current HP-71 BASIC settings.
- Certain flag conditions are established (refer to table 2.1). Flags that are matched to HP-71 system flags and user flags maintain that correspondence during operation of the emulator. For example, clearing HP-41 flag 26 to disable HP-41 audio functions also sets HP-71 flag −2. The emulator matches HP-41 and HP-71 user flags 0 through 7, allowing the HP-41 flags to control the HP-71 display annunciators.

Table 2-1. Flag Conditions At Startup

HP-41 Flag Number	Flag Name	Flag Condition
0 through 7	General purpose user flags	Matched to HP-71 user flags 0 through 7
11	Automatic Execution	Cleared
22	Numeric Data Input	Cleared
23	Alpha Data Input	Cleared
24	Range Error Ignore	Cleared
25	Error Ignore	Cleared
26	Audio Enable	Matched to opposite of HP-71 flag −2
27	User Keyboard	Matched to flag −9
36 through 39	Number of digits	Set according to display mode
42	GRAD Mode	Cleared if entering from BASIC, unchanged if entering from FORTH
43	Angular Mode	Matched to HP-71; set for RAD, clear for DEG
44	Continuous On	Matched to HP-71 flag −3
48	Alpha Keyboard	Cleared
50	Message	Cleared
52	Program Running	Cleared

- The HP-71 math exception traps are set as follows:

Table 2-2. Math Exception Traps

Trap	Value at Startup
Invalid operation	0
Division by zero	0
Overflow	0
Underflow	1
Inexact result	1

- The HP-IL system, if present, is searched for a printer device. If one or more printers are found, the printer at the lowest HP-IL device is selected as the output device for VIEW, AVIEW, and the HP-41 printer functions. Flags 21 and 55 are set if a printer is present; they are cleared otherwise. If you connect a printer on HP-IL after entering the HP-41 environment, you can activate the printer functions by typing PRINTER [END LINE].

If you are entering the HP-41 environment for the first time, you must perform one additional initialization procedure, described next.

Initializing the HP-41 Emulator

Before you can begin calculations with the HP-41 emulator, you must instruct the HP-71 to reserve a portion of its memory for use as HP-41 data registers. You will do this the first time you enter the HP-41 environment, and again if you have initialized the system (using the purge41 command) and then re-enter the HP-41 environment.

After you type HP41 to begin using the HP-41 emulator, the message:

```
HP-41 EMULATOR 1A
```

appears. If the system has not yet been initialized, it displays the prompt:

```
SIZE( max. nnn )?
```

This prompt instructs you to enter the number of data registers you intend to use initially with your HP-41 emulator. The number you enter is interpreted the same way as the parameter of the HP-41 SIZE function; that is, the SIZE selected is one more than the greatest register number—SIZE 100 corresponds to registers 0 through 99, and so forth.

The number *nnn* shown in the prompt is the number of registers that will fit in available HP-71 memory; each HP-41 emulator data register requires 8 bytes of user memory. There are two ways to specify the number of registers:

- Key in the number and press [END LINE]. You may enter any number from 0 through 10000. If your number is greater than *nnn*, *nnn* registers will be created.
- Press [END LINE] without a number. A default size of up to 319 registers (a maximum of 319 registers are available with the HP-41CV/CX) will be created. If *nnn* is less than 319, then *nnn* registers will be created.

After you have entered the SIZE, the HP-71 displays the current contents of the X-register, and is ready to begin accepting HP-41 commands.

Exiting the HP-41 Environment

Once you have entered the HP-41 environment, you can exit to BASIC or FORTH by typing `BASIC` [END LINE] or `FORTH` [END LINE]. In general, your HP-41 programs and registers are preserved intact for subsequent use when you re-enter the environment. It is, however, possible for you to alter or destroy HP-41 programs and data by using FORTH memory access words or the BASIC function `POKE`. Remember also that BASIC and the HP-41 emulator share certain user flags, and display and angular modes, so that changing any of these in one environment will carry over into the other.

The special HP-41 emulator function `purge41` is provided so that you can reclaim the HP-71 memory used by HP-41 programs and registers. Typing `purge41` [END LINE] in the HP-41 environment causes the system to enter the FORTH environment and to purge all HP-41 programs and registers. In addition, any FORTH words created since the HP-41 environment was initialized are purged.

If you turn the HP-71 off while in the HP-41 environment, it remains in that environment when you turn it on again. You can turn off the HP-71 either by pressing [f][OFF], or by executing the HP-41 `OFF` function. If you set flag 11 (automatic execution) and turn the HP-71 off using the `OFF` function, program execution begins at the current program pointer position when you turn the HP-71 back on.

Saving and Duplicating the HP-41 Environment

The HP-41 programs and data registers, together with the FORTH dictionary, are contained in a special HP-71 file (of type FORTH) named `FTH41RAM`. This file is created automatically, if it is not already present, when you enter either the HP-41 or FORTH environments from BASIC. From BASIC, you can treat the `FTH41RAM` file just as you would any other file, using any of the HP-71 file commands. By renaming `FTH41RAM` (using the BASIC `RENAME` statement), you can archive a particular version of the file in main memory, independent RAM, or on a mass storage medium. If you rename `FTH41RAM`, the system automatically creates a new file named `FTH41RAM` when you enter the HP-41 environment.

The file currently named `FTH41RAM` is the active file when you enter the FORTH or the HP-41 environments.

HP-41 Keyboard Calculations

The HP-41 Translator Pac allows you to run programs written in HP-41 user language on your HP-71. Since many HP-41 programs require you to perform keyboard calculations as part of their operation, the HP-41 Translator Pac includes an HP-41 emulator that provides keyboard calculation capability. However, the emulator does not provide an exact, keystroke-for-keystroke imitiation of the HP-41. Instead, the emulator provides a near copy of the HP-41 user interface, close enough to the original to be easy to learn, yet changing some features and adding others to take advantage of the strengths of the HP-71. The next several subsections discuss the major similiarities and differences between the actual HP-41 and the emulator.

To begin with, let's summarize the HP-41 memory features that are present in the emulator:

- An RPN arithmetic stack, with 4 working levels named X, Y, Z, and T, plus a LAST X register.
- A 24-character alpha register.
- A user-specified number of fixed data registers, each of which can store one floating-point number or one 6-character alpha string.
- 30 user flags, numbered 0 through 29, plus additional HP-41 system flags numbered 31 through 55, that can provide status information to programs.
- Program memory. The amount of memory used for HP-41 programs varies as you load or clear programs. All memory packing is carried out automatically by the emulator.

Keyboard Differences

The most obvious difference between the HP-41 and the HP-71 is their keyboards, which differ in the number and layout of their keys and in the basic style of user input. The HP-41 is optimized for key-per-function operation; functions can be executed with a single keystroke. Any function not present on the keyboard can be assigned to a key. The alpha keyboard, which can be used for spelling out functions not assigned to keys or for alpha register entry, is available only through a mode key, which disables the normal keyboard.

The HP-71 keyboard, on the other hand, is a "block-querty" keyboard plus number pad and typing aids; the keyboard has no default key-per-function keys. In the BASIC environment, you must spell out a command or expression, then press [END LINE]. The [END LINE] key acts as a command terminator. When you press it, the HP-71 processes the instruction(s) you have typed. Each line you type in must follow strict syntax rules so that the BASIC interpreter can understand it.

Executing HP-41 Functions

The HP-41 emulator offers an input style intermediate between HP-71 BASIC and the actual HP-41. Like the HP-41, any function can be spelled out and executed by name without any special syntax. Unlike the HP-41, however, the only built-in dedicated function key is [RUN]. For example, suppose you wish to take the sine of the number in the X-register. On the HP-41 you press the [SIN] key to see the result in X. On the HP-71, you must type out [S][I][N][END LINE]*. As you type, the display echoes each character so that you can see what you enter. Up to the final [END LINE], you can use any of the HP-71 editing keys to change your entry. After you press [END LINE], your entry is processed and you see the result in the display.

The HP-41 emulator style of function execution is analogous to the HP-41 method of executing non-keyboard functions. For example, to execute the MOD function on the HP-41, you must press [XEQ][ALPHA][M][O][D][ALPHA]. The final [ALPHA] keystroke acts like the HP-71 [END LINE], telling the calculator to execute the function you have spelled out. But notice that on the HP-71, the initial [XEQ][ALPHA] sequence is unnecessary; you just type [M][O][D][END LINE].

The emulator is case sensitive—functions must be spelled out in uppercase letters. The function purge41 is the only emulator function that uses lowercase letters.

Emulator Display Functions

Two functions are provided in the emulator that alter the normal X-register display. The new display mode remains in effect until a new HP-41 emulator display function is executed.

Displaying the entire stack. The word STACK causes the HP-71 to display the entire RPN stack, including the LAST X register. The stack registers are displayed left-to-right across the display, in the order T, Z, Y, X, and L. LAST X is separated from X by a vertical bar |. Numbers are shown in the current display format. For example, typing:

```
FIX 2 STACK [END LINE]
CLST [END LINE]
2 3 4 [+] 5 [*] [END LINE]
```

displays the register contents:

```
0.00 0.00 2.00 35.00 | 5.00
```

When the display exceeds 22 characters, it scrolls at the rate determined by the current HP-71 delay setting. After the scrolling is complete, you can view any portion of the display using the left and right cursor keys.

Displaying the alpha register. The A,X function changes the display mode to display the alpha register and the X-register, separated by vertical bar, after each HP-41 command sequence.

* Like the HP-41, the HP-71 can assign any function to a key for single keystroke operation. For example, the character sequence SIN can be assigned to ■ key. The KEYS41 file, discussed on pages 28 through 30, contains ■ number of assignments.

Displaying the X-register only. The function XONLY restores the standard X-register display mode.

Replacing the Σ, ≠, and ⊢ Characters

The HP-71 keyboard lacks the HP-41 characters Σ, ≠, and ⊢. Although all three characters are in the HP-71 character set, there are no key sequences that produce the characters unless you make special key assignments. Rather than requiring you to dedicate keys for this purpose, the HP-41 Translator Pac renames all of the HP-41 functions containing these characters:

Table 2-3. Replacements for Σ and ≠

Emulator function	HP-41 function	Meaning
S+	Σ+	Summation plus
S-	Σ−	Summation -
CLS	CLΣ	Clear statistics registers
SREG	ΣREG	Select statistics registers
SREG?	ΣREG?	Identify statistics registers
X#Y?	X≠Y?	*x* not equal to *y*?
X#0?	X≠0?	*x* not equal to 0?
X#NN?	X≠NN?	*x* not equal to contents of register *nn*?

Also, the HP-71 symbol › is used in place of the HP-41 append symbol, ⊢, in alpha entry program lines.

Entering Numbers

Number entry differs significantly between the HP-41 and the HP-41 emulator. When you press the first number key on the HP-41, you initiate a special number entry sequence. This entry sequence includes the _ prompt to indicate the next digit position, automatic digit separators (, or .), and the ability to change the sign of the number at any time (before you've pressed [EEX]) by pressing [CHS]. [EEX] starts exponent entry; you can change the sign of the exponent any time during exponent entry with [CHS].

The HP-41 emulator requires you to enter numbers by typing them exactly as they might be displayed, terminating entry with [END LINE]. There is no [CHS] key; to enter a negative number, you must precede it with a leading minus sign or execute CHS after entering the number. You cannot use spaces or other digit separators within a number.

To add an exponent to a number after typing the mantissa (with as many digits as you wish—the HP-71 will keep a maximum of 12), type the letter E, followed by − or an optional +, then up to three exponent digits. The exponent you enter must fall in the range −499 through +499. All numbers must have a mantissa. For example, 10^5 must be entered as 1E5.

Examples:

Typed Entry	Display (assuming FIX 4)
123456 [END LINE]	123456.0000
1.234 [END LINE]	1.2340
123.4566 [END LINE]	123.4566
1E25 [END LINE]	1.0000E25
10.5E+456 [END LINE]	1.0500E457
-5.3E-1 [END LINE]	-0.5300
-227 [END LINE]	-227.0000
1,234.00 [END LINE]	ERR:Data Type (No digit separators allowed)
E25 [END LINE]	ERR:Data Type (Mantissa missing)

Multiple Entries

The HP-41 emulator accepts any sequence of functions and numbers keyed in together, separated by spaces and terminated by a final [END LINE]. For example, suppose you wish to evaluate the expression:

$$\text{SIN}\left(\frac{1.234 \times 2}{\text{COS}(31.9)}\right)$$

On the HP-41, you would press the following keys:

HP-41 Emulator Keystrokes	Display	Meaning
2 [ENTER↑]	2.0000	Number entry
31.9 [COS]	0.8490	COS of 31.9°
[÷]	2.3558	2/.8490
1.234 [×]	2.9070	2.3558 × 1.234
[SIN]	0.0507	SIN of 2.9070°

On the HP-41 emulator, you can choose to perform the arithmetic operations one at a time, or all at once. The following keystrokes allow you to see each intermediate step in the calculation:

HP-41 Emulator Keystrokes	Display	Meaning
2 [END LINE]	2.0000	Number entry
31.9 [END LINE]	31.9000	Number entry
COS [END LINE]	0.8490	COS of 31.9°
/ [END LINE]	2.3558	2.0000/.8490
1.234 [END LINE]	1.2340	Number entry
* [END LINE]	2.9070	2.3558 × 1.2340
SIN [END LINE]	0.0507	SIN of 2.9070°

If you are interested only in the final result, you can key in the entire sequence on one line, using spaces to separate entries.

```
2 31.9 COS / 1.234 * SIN [END LINE]
```

When you press [END LINE], the final result 0.0507 is displayed.

Each sequence can be up to 96 characters long. Each entry in the sequence is processed from left to right. When the sequence is complete, the final result is displayed. The entire sequence is also stored in the HP-71 command stack, where it can be edited and re-executed.

> **Note:** To enter the command stack, press [↑] or [↓]. You do not need to press [9][CMDS] first. If you have assigned the arrow keys to other functions, you will have to disable USER mode or use [9][1 USER] to enter the command stack.

Evaluating Input

The emulator breaks up an input sequence into separate entries, delimited by spaces. The emulator then analyzes each entry, returning to step 1 each time an entry is successfully evaluated:

1. First, it tries to find the entry in the HP-41 and FORTH function lists. If an entry is found, the function is executed.
2. If an entry is not found in the HP-41 or FORTH function lists, it is sent to the BASIC environment for possible evaluation as a number or numeric expression. If the entry is a BASIC numeric expression, it is evaluated and entered into the X-register, lifting the RPN stack.
3. If the input is not a valid BASIC expression, the BASIC error message `ERR:Data Type` is displayed, and entry analysis ceases.

When an error occurs, any pending functions or numbers remaining in the input sequence are lost.

Emulator Stack Lift

The most important difference between the HP-41 emulator and the original HP-41 is the absence of *stack lift disable* in the emulator during keyboard entry. (Stack lift disable *is* implemented while programs are running.) This absence does not effect the calculation capabilities of the emulator, but it does represent a significant change for the experienced RPN calculator user.

The HP-41 and other RPN calculators do not have a special key used to terminate number entry. When you key a number into the X-register, the digit entry prompt _ remains in the display until you press a non-numeric key. Typically, this non-numeric key is an arithmetic operator (for example, [+]), a function key which operates on the new number in X (for example, [SIN], [1/x]), or the [ENTER↑] key. The [ENTER↑] key is used to separate consecutive numbers that you enter into the RPN stack. It has three purposes—it terminates number entry, copies the number into the Y-register, and disables stack lift. With stack lift disabled, the next number you enter overwrites the X-register without lifting the stack.

The HP-41 emulator always has stack lift enabled. The stack is lifted before new data is entered into the X-register by digit entry or by a function (for example, RCL, TIME, PI). Digit entry is always terminated by either [SPC] or [END LINE]. You use [SPC] when you wish to key in additional numbers or functions as a multiple entry. Neither [SPC] nor [END LINE] copies the contents of the X-register into the Y-register; each successive number always raises the stack. To copy the contents of the X-register into the Y-register, use the RCL X or ENTER^ function.

Here are a few simple examples that illustrate the two methods of number entry:

HP-41 Keystrokes	HP-41 Emulator Input Sequence	Result
1 [ENTER↑] 2 [+]	1 2 + [END LINE]	3.0000
3 [ENTER↑] 5 [ENTER↑] 2 [+] [/]	3 5 2 + / [END LINE]	0.4286
30 [SIN] 2 [+] [ENTER↑] [+]	30 SIN 2 + ENTER^ + [END LINE]	5.0000

The other HP-41 functions that disable stack lift, [CLX], [Σ+], and [Σ−], are similarly altered in the emulator. On the HP-71, CLX replaces the contents of the X-register with zero, but does not disable stack lift. S+ and S− also perform identically to their HP-41 counterparts except for the stack lift disable.

The following table summarizes replacements for [ENTER↑] and [CLX]:

Table 2-4. Replacements for [ENTER↑] and [CLX]

HP-41 Key	HP-41 Emulator Function
[ENTER↑]	• To separate two numbers, use [SPC] or [END LINE]. • To copy the X-register into the Y-register, use ENTER^* or RCL X.
[CLX]	• To change the number in the X-register, use RDN, then key in the new number. • To enter 0 into the X-register, use 0 or CLX.

* Be sure to include the final ^ character when you spell out ENTER^, since the ^ character differentiates the function from the FORTH HP-IL word ENTER.

The absence of stack lift disable is important only for HP-41 emulator keyboard calculations. The stack lift disabling functions behave in emulator programs exactly as they do on the HP-41. When you write HP-41 programs on the HP-71, or transfer programs from the HP-41 to the HP-71, the Pac's translator automatically provides appropriate stack lift enable or disable within the translated program. Therefore, you can run HP-41 programs exactly as they are written on the HP-41.

Two-Part Functions

Certain HP-41 functions require you to specify a *parameter*—a number following the function name that specifies a register number, a flag number, a tone number, or a number of display digits. To execute any of these functions, type the function exactly as it would be displayed in an HP-41 program. That is, you type the name of the function, a space, and the parameter number or letter. You can press [END LINE] to execute the function immediately, or you can type another space and continue with a series of commands or numbers. Here are some examples:

HP-41 Keystrokes	HP-41 Emulator Input Sequence
[STO] 25	STO 25 [END LINE]
[XEQ] [ALPHA] FIX [ALPHA] 5	FIX 5 [END LINE]
[RCL] [.] [L]	RCL L [END LINE]
[XEQ] [ALPHA] X [■] < [■] > [ALPHA] 98	X<> 98 [END LINE]*
[XEQ] [ALPHA] FIX [ALPHA] [■] 20	FIX IND 20 [END LINE] *or* FIX -21 [END LINE] (negative parameter, see discussion of indirect addressing, below)
[RCL] [■] [.] [X]	RCL IND X [END LINE]

There are several differences between the HP-41 keystrokes and the HP-41 emulator input sequences:

- There is no need to type a leading zero when a numeric parameter is a single digit.
- On the HP-41 emulator, you are not limited to register numbers in the range 0 through 99.
- You must be certain to type a space between the function name and its parameter.
- Indirect addressing can be specified by typing IND before the parameter. If the parameter is a number, indirect addressing can also be specified by using a negative parameter as follows:

  $$\text{parameter} = -(\text{register number} + 1)$$

  For example, STO -1 is equivalent to STO IND 0, and VIEW -4 is equivalent to VIEW IND 3.

In either the direct or indirect cases, you must type the entire function in a single line of input; you cannot press [END LINE] between parts of the function. If you type a function name (or the name plus IND) without a numeric parameter and press [END LINE], the emulator ignores the input.

Simulating HP-41 Keystrokes With the USER Keyboard

The HP-41 emulator can simulate HP-41 keystrokes using HP-71 key assignments provided by a built-in key file named KEYS41. KEYS41 contains key assignments that enable the HP-71 keyboard to emulate most of the HP-41 keyboard.

* Spacing here is critical. X<> 98 exchanges the contents of the X-register with the contents of R_{98}. X<>98, however, is evaluated as a BASIC boolean expression, returning 0 if variable X has been assigned the value 98, 1 otherwise.

Note: To illustrate how key definitions emulate the HP-41, try redefining the [+] key to work like its HP-41 counterpart. From the BASIC environment (type `BASIC` [END LINE] if the HP-41 emulator is active), type:

```
KEY "+"," +" [END LINE].
```

Be sure to include the space in front of the second + in the expression.

Return to the HP-41 environment by typing `HP-41` [END LINE], and press [f][USER] to activate user mode. Now type:

```
2 [END LINE]
3+
```

When you press [+], you briefly see `3 +` in the display before the system displays the answer, `5.0000`.

Using the HP-71 `KEY` statement, you can assign any HP-41 function or input sequence to any HP-71 key. If you use an immediate-execute assignment as in the preceding example, you should include a leading space in your key definition so that you don't have to type a space before pressing the assigned key.

To activate the `KEYS41` assignments, type `KEYS41`. This HP-41 emulator function merges the `KEYS41` file included in the HP-41 Translator Pac with any existing keys file.

To activate the user keyboard, press [f][USER]. The emulator is now in USER mode. Table 2-5 lists the key assignments made by `KEYS41` that become active in USER mode.

Table 2-5. HP-41 Emulator User Key Assignments

HP-41 Key	HP-71 Key	HP-41 Key	HP-71 Key	HP-41 Key	HP-71 Key	HP-41 Key	HP-71 Key
[÷]	[/]	[XEQ]	[f][W] (alpha) [f][S] (numeric)	[RTN]	[f][D]	[1/x]	[f][M]
[×]	[*]	[GTO]	[f][R] (alpha) [f][F] (numeric)	[X≤Y]	[f][=]	[e^x]	[f][(]
[−]	[−]	SF	[f][Y]	FS?	[f][H]	[LN]	[f][)]
[+]	[+]	CF	[f][U]	FC?	[f][J]	[SIN^{-1}]	[f][1]
[−] (simple minus)	[.]	[Σ+]	[f][0]	[BEEP]	[f][L]	[COS^{-1}]	[f][2]
[RCL]	[↑]	[Σ−]	[f][P]	[SIN]	[f][4]	[TAN^{-1}]	[f][3]
[STO]	[↓]	MEAN	[f][8]	[COS]	[f][5]	[LOG]	[f][−]
R↑	[g][↑]	SDEV	[f][9]	[TAN]	[f][6]	[VIEW]	[f][.]
[R↓]	[g][↓]	[√x]	[f][/]	[CLΣ]	[f][7]	[LASTx]	[f][+]
[ALPHA]	[f][RES]			[CATALOG]	[f][X]	[10^x]	[f][*]

The overlay provided with the HP-41 Translator Pac labels the USER mode key assignments. The key assignments are color-coded according to the following conventions:

- To access functions printed in grey, press the unshifted key **below** the label.
- To access functions printed in yellow, press [f]-shifted key **below** the label.
- To access functions printed in blue, press the [g]-shifted key **above** the label.
- Red labels (BACK and APPEND) indicate these assignments are active only in ALPHA mode. To access these functions, press the unshifted key **below** the label.

The following general rules apply to the USER mode keyboard:

- The arithmetic operators keys [+], [−], [*], and [/] are assigned as immediate-execute keys.
- Since the [−] key is redefined as an immediate-execute key, it cannot be used to enter negative numbers. Therefore, the [,] key is redefined as a simple minus sign. To enter a negative number, type the simple minus before typing the number.
- The ALPHA mode key assignments are part of the emulator itself. They are always active in the emulator ALPHA mode, regardless of whether KEYS41 is the current keys file. USER mode is disabled when you enter ALPHA mode and reenabled when you exit ALPHA mode.
- In addition to the four ALPHA mode assignments shown on the overlay (BACK, APPEND, CLA, and ASHF), the [↑] key (labeled RCL) executes ARCL, and the [↓] key (labeled STO) executes ASTO.
- The XEQ" " and GTO" " assignments actually enter XEQ"" and GTO"", with the insert cursor pointing between the quotes. To execute or go to an alpha label, type the name of the label and press [END LINE]. For example, to execute alpha label STEST, press [f][W] STEST [END LINE].

The following keystrokes evaluate the numeric expression:

```
SIN(1.234*2/COS(-31.9))
```

HP-41 Emulator Keystrokes	Display
2 [END LINE]	2.0000
[,] 31.9 [COS] (type [f][5] for [COS])	0.8490
[/]	2.3558
1.234 [*]	2.9070
[SIN] (type [f][4] for [SIN])	0.0507

Evaluating BASIC Numeric Expressions

The HP-41 emulator takes advantage of the HP-71 BASIC operating system to provide a feature that has no counterpart in the HP-41: you can evaluate algebraic expressions directly without translating the expression to RPN keystrokes. For example, consider the expression SIN(1.234*2/COS(-31.9)) that we evaluated previously in RPN format. You can type in this expression exactly as it is written in BASIC algebraic format, then press [ENDLINE]. The emulator evaluates the expression and enters the result into the X-register. Any legal BASIC expression is acceptable, as long as it contains no spaces. The expression can contain BASIC numeric variables, and any numeric function recognized by the HP-71, language extension files, or plug-in modules.

Examples: If the USER annunciator is on, press [f][USER] to exit USER mode before continuing.

HP-41 Emulator Keystrokes	Display
1+2+3+4 [END LINE]	10.0000
ASIN(0)	0.0000
PI>3	1.0000 (true)
sqr(log(fact(10)))	3.8864

Uppercase and lowercase letters can be used interchangeably in BASIC functions.

Using BASIC Variables

The HP-41 emulator can assign values to BASIC numeric variables and recall BASIC variables into the X-register. If two or more variables are recalled at the same time, the values are lifted into the other stack registers.

Recalling BASIC Variables. To recall the value of a BASIC variable into the X-register, type the name of the variable. For example, typing:

A [END LINE]

recalls the value of the BASIC variable A to the X-register. Typing:

A B C D1 [END LINE]

recalls the value of the BASIC variable A to the T-register, B to the Z-register, C to the Y-register, and D1 to the X-register.

Assigning Values to BASIC Variables. To store the contents of the X-register into a BASIC variable, type the name of the variable preceded by an exclamation point. For example:

45.6789 !G [END LINE]

places 45.6789 in the X-register and then assigns that value to variable G. The assignment can be done in one step; for example:

1234 !Q9 [END LINE]

assigns the value 1234 to variable Q9. Two or more assignments can be made on the same line. Thus:

1234 !Q9 4567 !R [END LINE]

assigns the value 1234 to Q9, and 4567 to R.

You can use additional features of the HP-71 BASIC environment by using the FORTH words BASICX and BASICF. Refer to Appendix D.

Alpha Mode

The HP-41 emulator provides a 24-character alpha register and the full set of HP-41 alpha functions. Keyboard operations with the alpha register are virtually identical to those on the HP-41, so you only need to learn which HP-71 keys match the HP-41 alpha keyboard.

Entering and Leaving Alpha Mode.. The emulator uses the ‡ ([g][4]) symbol to activate ALPHA mode. (The ‡ symbol is used in BASIC to identify string variables and functions, hence its choice for the HP-41 emulator [ALPHA] key). There are three ways to enter ALPHA mode:

- Type ‡ [END LINE]
- If KEYS41 is the active keys file, press [f][RES] (labeled AON on the overlay).
- Type AON [END LINE].

The AC annuciator indicates that the system is in ALPHA mode. The current contents of the alpha register is displayed. If the register is empty, the display is blank.

To exit ALPHA mode at any time, press [END LINE].

The Alpha Keyboard. When ALPHA mode is active, the HP-71 alphanumeric keys act to append the appropriate character to the current alpha register string. The cursor keys assume the roles of the special HP-41 alpha keys. The correspondence is shown in table 2-6.

Table 2-6. Special Emulator Alpha Keys

HP-71 Key	HP-41 Key	Function
[→]	append	Begin append; turn on the _ prompt.
[←]	[←]	Delete the last character (or clear alpha register if append prompt is absent).
[g][←]	■[←]	Clear alpha register (CLA).
[g][→]	none	Perform ASHF without exiting alpha mode.
[↑]	ARCL	Recall alpha data.
[↓]	ASTO	Store alpha data.
[RUN]	[R/S]	Run program.
[ENDLINE]	[ALPHA]	Exit alpha mode to execution mode.

Any key not listed in the table just adds a character to the alpha string. ([ATTN], for example, displays the character ↑; [g][↑] displays π; [g][↓] displays Ä). Characters associated with the special alpha keys listed in the Table 2-6 are not available. All other HP-71 characters except control characters are available in ALPHA mode, including symbols and lowercase letters. The keys associated with special characters are listed on page 42 of the HP-71 Quick Reference Guide. Since the append key [→] is only useful when the append prompt is absent, it generates the σ character when the prompt is already present.

If the append prompt _ is present, a new character is appended to the current string, and the [←] key deletes the last character in the register.

If the append prompt is absent, pressing a character key replaces the current string with a new character and turns on the prompt. The [→] key turns on the append prompt without clearing the alpha register, and the [←] key clears the alpha register.

Certain HP-41 display characters are unique to that calculator, but are replaced on the HP-71 by ASCII characters with the same character code but different appearance. For example, character codes 2, 3, 7 through 11, 14 through 28, 30, 31, and 102 through 125 are displayed as the "starburst" character of the HP-41, but appear as distinct characters on the HP-71. Character code 0, which is displayed as an overline on the HP-41, is invisible on the HP-71. In general, the numeric data associated with alpha characters (obtained with ATOX, POSA, etc.) are the same on both calculators.*

Storing and Retrieving Alpha Data. ASTO and ARCL can be executed from ALPHA mode in much the same way as two-part functions are executed in execute mode. When you press [↓] in ALPHA mode, ASTO appears in the display. You must type in a register number or letter, which can be preceded by IND, and then press [END LINE]. The normal alpha register display is then restored with the append prompt off.

ARCL ([↑]) appends the contents of the specified register to the existing contents of the alpha register. At the end of the ARCL operation, the append prompt is on.

You can enter characters into the alpha register without activating ALPHA mode. In execute mode, just type the desired alpha string surrounded by quotes. If the first character after the leading quote mark is ⊢ or ⟩, the rest of the string is appended to the current alpha register contents. For example,

```
"ABCD" "⟩EFGH" [END LINE]
```

puts the string ABCDEFGH into the alpha register.

> **Note:** You cannot have a space as the first character of a quoted string for keyboard alpha register entry. A quote followed by a space is a FORTH function (refer to Section 5).

* When programs are transferred from the HP-41 to the HP-71 via HP-IL using READ41 (described in Section 3), character codes 10 (♦), 13 (<), and 126 (Σ) are automatically replaced by codes 0 (null character), 124 (|), and 28 (Σ), respectively.

Program Execution

Running Programs

There are three ways to run a program:

- Press [RUN].
- Type RUN [END LINE].
- Type XEQ *label number* [END LINE] or XEQ"*alpha label*" [ENDLINE].

In addition, you can use GTO, END, or RTN to place the HP-41 program pointer at program labels or the start of the program prior to execution. In ALPHA mode, you can begin program execution by pressing the [RUN] key.

The following table summarizes the implementation of HP-41 program execution functions on the HP-41 emulator.

Table 2-7. Program Execution on the HP-41 Emulator

HP-41 Keystrokes	HP-41 Emulator Function and/or Key	Result
[R/S]	RUN or [RUN]	Begins program execution at the current program pointer position.
[XEQ]	XEQ *label number*	Begins program execution at the specified numeric label.
[XEQ] ■ *register number*	XEQ IND *register number*	Begins program execution at the label specified in a register.
[XEQ] [ALPHA] *label* [ALPHA]	XEQ"*alpha label*"	Begins program execution at the specified alpha label.
[GTO]	GTO *label number*	Moves the program pointer to the specified numeric label.
[GTO] ■ *register number*	GTO IND *register number*	Moves the program pointer to the label specified in the register.
[GTO] [ALPHA] *label* [ALPHA]	GTO"*alpha label*"	Moves the program pointer to the specified alpha label.
[RTN]	RTN	Moves the program pointer to the start of the current program.
[R/S]	[ATTN]	Halts the program. Use [RUN] to continue execution.

Notice that alpha GTO and XEQ follow the syntax of HP-41 program listings. You can think of the quotation marks surrounding the function name as representing the HP-41 [ALPHA] key. To execute a program on the HP-41 at LBL"BETA", you press [XEQ] [ALPHA] BETA [ALPHA]. In the emulator, you type XEQ"BETA", substituting quotes for the [ALPHA] key.

If you execute RUN when there are no HP-41 programs present in HP-41 emulator memory, or after you have cleared a program using CLP, the HP-71 displays the error message No program selected. After clearing a program, you must use an alpha XEQ, GTO, or CAT to position the program pointer within an existing program before you can use RUN.

HP-41 programs loaded into HP-41 emulator memory are actually compiled FORTH words. In this form, there are no program lines as such, so that you cannot move the program pointer to a specific HP-41 program line number, nor can you single-step through a program.

Pausing a Program From the Keyboard

Pressing [ATTN] while an emulator program is running halts program execution the next time one of the following functions is encountered:

```
AVIEW   PROMPT   STOP
END     PSE      VIEW
GOTO    RTN      XEQ
```

The function is not executed before the program halts. Pressing [RUN] resumes execution at that function.

If a program contains none of these functions other than a final END, the program cannot be halted until execution is completed.

The INIT 1 and INIT 2 commands also stop program execution. However, the data in the HP-41 emulator registers may be incorrect. Pressing [RUN] after an INIT 1 or INIT 2 resumes program execution, but not necessarily at the place where the program was halted.

Program Halts and Pauses

There are five HP-41 functions included in the emulator that halt or pause program execution, and permit you to resume execution at your option. STOP, PROMPT, AVIEW and VIEW operate exactly as they do on the HP-41 (VIEW and AVIEW halt execution only if Flag 21 is set and Flag 55 is clear). However, PSE differs from its HP-41 counterpart in one important respect.

The HP-41 PSE function suspends execution for one second, at which time you can enter numeric or alpha data. Each press of an alpha key or numeric key extends the pause for one second. If you press any other key, the program halts. The HP-41 emulator PSE function also suspends execution for one second, but any key pressed will halt program execution, after which you must use RUN to resume.

Program Errors

The HP-41 emulator responds to errors using a combination of HP-41 error detection and HP-71 error handling and reporting. In general, an error causes the HP-71 to beep and display an error message. If flag 25 is clear, an error halts the program and resets the program pointer to the beginning of the current program. If flag 25 is set, the program function causing the error is skipped. Program execution continues (with no error message) and flag 25 is cleared.

The HP-41 emulator can generate six HP-41 error messages:

```
Alpha Error
Data Error
Nonexistent
No Printer
Insufficient Memory
Not Programmable
```

These messages are always prefixed by FTH ERR:, indicating that the error originated within the FORTH system.

In addition to these standard HP-41 error messages, some emulator functions can also generate more specific and informative HP-71 BASIC errors. For example, executing LOG with −1 in the X-register returns the error:

```
ERR:LOG(neg)
```

instead of the Data Error message used by the HP-41 emulator. (BASIC system error messages are prefixed by ERR:.) Refer to pages 378 through 381 of the HP-71 Reference Manual for a complete list of BASIC errors.

All emulator errors, except those caused by non-programmable keyboard operations, update HP-71 values returned by the BASIC functions ERRN and ERRM$. The keyboard operations that do not affect ERRN and ERRM$ are:

- Executing ASTO or ARCL in ALPHA mode to a non-existent register, returning the Nonexistent error.
- Attempting to execute RUN when the program pointer is not initialized, returning the No Program Selected error. If this error is generated in ALPHA mode, the emulator leaves ALPHA mode.
- Executing CAT when emulator memory is empty, returning the No 41 Programs error.

Clearing Programs

The CLP function removes an HP-41 program from emulator memory. The syntax is:

CLP"*alpha label*"

where the alpha label is contained in the program you wish to delete. Note, however, that the HP-41 emulator CLP function works more like the HP-41 function PCLPS than the non-programmable CLP; that is, it clears not only the specified program, but also all programs that were loaded after the named program.*

When you execute CLP, the system displays PACKING momentarily to remind you that the memory used by the cleared programs is being restored for general use.

Cataloging Programs

The HP-41 emulator function CAT works similarly to the HP-41 CAT 1 function by listing the alpha labels contained in programs in emulator memory. There are two major differences:

- CAT does not list program ENDs.
- CAT lists the alpha labels in last-to-first order, the opposite of the HP-41 CAT 1 function.

CAT displays each label for approximately 1 second. Pressing any key except [ATTN] while a label is showing moves the program pointer to the label in program memory and then terminates the catalog. Pressing [ATTN] terminates the catalog without moving the program pointer.

If there are no HP-41 programs in emulator memory, CAT displays No 41 Programs.

* Any FORTH words compiled after the named HP-41 program will also be cleared from the FORTH dictionary by CLP.

Section 3

Writing and Translating HP-41 Programs

Introduction

This section describes:

- How to write HP-41 user-language programs using the HP-71 text editor, and how to load those programs into the HP-41 emulator memory.
- The procedure for transferring an HP-41 program directly (via HP-IL, requiring HP-IL modules for both calculators) from HP-41 memory to the HP-71 file system, and then to emulator memory.
- How you can enhance the HP-41 user-language by creating your own new HP-41 functions.

The Translation Process

Converting an HP-41 program from its original HP-41 user-language form to a form that can be run by the HP-41 emulator is a 3-step process. Although the programs READ41 and TRANS41 provided in the HP-41 Translator Pac can carry out the entire process automatically, you should understand the purpose and results of each step in the process so that you can make optimum use of HP-71 memory and mass storage.

Step 1, performed in the BASIC environment, is creating an HP-41 user-language program in an HP-71 text file. You can either use the HP-71 text editor included in the pac to write the program from scratch or copy it from an HP-41 program listing, or you can use READ41 to transfer a program directly from HP-41 memory to an HP-71 text file, via HP-IL. Once the program is in a file, you can save it on a mass storage device or on magnetic cards. At this stage, the program text looks very similar to an HP-41 printed program listing.

Step 2 is translating the contents of the HP-41 user-language text file into a form that is suitable for loading into the HP-41 emulator memory, using the program TRANS41. In this intermediate form, the program is still contained in a text file that you can view or list using the text editor. However, this version of the program contains automatic memory management instructions added by TRANS41, plus modifications or replacements of certain HP-41 program lines. You should not try to edit the program in an intermediate file. But you can store intermediate files on mass storage or magnetic tape, so that you can eliminate the translation step the next time you want to load the program into the emulator. Step 2 is also performed in the BASIC environment.

Step 3 can be performed from the BASIC environment, as part of TRANS41, or from the HP-41 environment, using the emulator function LOAD. This last step loads the intermediate program into emulator memory, where it can be run using the emulator functions XEQ or RUN. The emulator does not contain a program mode like that of the HP-41, so you cannot edit or list the program in its final form.

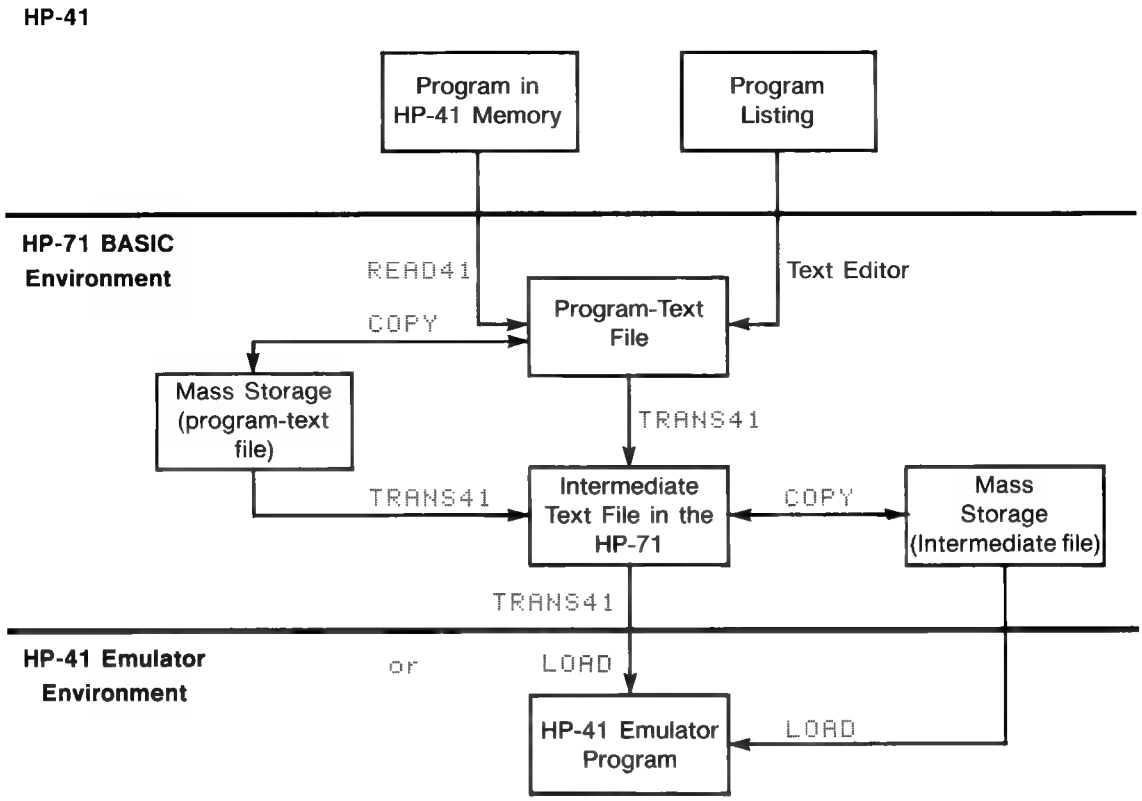


Figure 3-1. The Translation Process

Writing HP-41 Programs

You can write HP-41 programs directly on the HP-71 using the HP-71 Text Editor Program (editor, for short) included in the HP-41 Translator Pac. Using the editor, you can create new programs, much as you would on the HP-41, or you can copy programs directly from HP-41 program listings, matching the listings character-for-character, line-for-line.

The following simple HP-41 program adds 1 plus 2, and then uses the alpha register to display a labeled result. The program is shown as it would be output in an HP-41 printer listing.

```
01 ♦LBL"THREE"
02 1
03 ENTER↑
04 2
05 +
06 "RESULT="
07 FIX 2
08 ARCL X
09 AVIEW
10 END
```

The following procedure creates this program on the HP-71:

1. Enter the BASIC environment.
2. It is recommended that you PURGE or RENAME the current KEYS file before using the editor to remove all previous key assignments.
3. Enter the editor by typing:

   EDTEXT *new file name* [END LINE]

   using any legal HP-71 file name. This creates an HP-71 text file and runs the editor program. For example, to enter the editor for the purpose of creating a file containing program THREE, above, type:

   EDTEXT THREE [END LINE]

   After a brief pause while the editor initializes, it displays:

   Eof, Cmd:

   The Cmd: prompt indicates that the editor is awaiting a command. Eof (end-of-file) indicates that the file is empty.
4. Type T [END LINE] to put the editor into text entry mode. In text mode, the display always shows the current line in the file. Since the file is empty, the display is now blank.
5. Key in the program. For example, the previous program is entered by typing:

   LBL"THREE" [END LINE]
   1 [END LINE]
   ENTER^ [END LINE]
   2 [END LINE]
   + [END LINE]
   "RESULT="
   FIX 2 [END LINE]
   ARCL X [END LINE]
   AVIEW [END LINE]
   END [END LINE]

   After each [END LINE], the editor stores the line you have just typed in, and moves to the next (in this case, blank) line in the file.
6. Press [ATTN] to exit text entry mode and return the editor to command mode. The editor once again displays the Eof, Cmd: prompt.
7. Type E [END LINE], to terminate the editing session. The HP-71 displays:

   Done: *file name*

   At this point, you can store the file on magnetic cards or a mass storage device, or use the program TRANS41 to translate the program into its intermediate form.

The editor provides additional features, including inserting and deleting lines, search and replace, printer and display listings. These features are described in Section 4.

The translator program TRANS41 requires that each line of the text file contains only one HP-41 program line, and that each HP-41 program line be entirely contained on one line of the text file. As you are entering the program, therefore, always follow the rule that one text line equals one program line. In general, you should try to make your text file program listing look exactly like an HP-41 printer listing of the same program (assuming that the printer was listing in manual mode with flags 15 and 16 clear). Do not type in program line numbers, since the editor keeps track of line numbers for you. (Also, do not type in the ▪ character that precedes labels in printer listings on the HP 82143A and HP 82162A printers.) Some other points to keep in mind:

- HP-41 text strings in text lines or following GTO and XEQ should be enclosed in double quotes ("), as they are in HP-41 printer listings. One or more spaces between GTO or XEQ and the quoted string are optional.

  Examples:

  ```
  "ABCDE"
  GTO"FRED"
  XEQ "GEORGE"
  ```

- The Σ, ≠, and ⊢ characters, which are used frequently in HP-41 programs, are not available on the HP-71 keyboard. You can include these characters in your programs either by assigning HP-71 keys to these characters (Σ is CHR$(28), ≠ is CHR$(29), and ⊢ is CHR$(127)), or by substituting standard keyboard characters:
  - In conditional functions, substitute # for ≠.
  - In any HP-41 functions containing Σ, substitute the letter S for Σ. For example, Σ+ becomes S+.
  - In an HP-41 text line, you can substitute the character) for the ⊢ symbol immediately following the first quotes. For example, "⊢add this" becomes ")add this".
- HP-41 emulator programs must contain at least one global alpha label so that you can access the program in the HP-41 environment. If you do not include an alpha label, TRANS41 automatically adds a label to the beginning of the translated program. The label uses the name of the intermediate file for its alpha characters.
- Parameters for two-part functions must fall in the ranges:

  Registers: 0 through 9999
  Numeric local labels: 0 through 9999
  Local alpha labels: A through J; a through ▪
  Display formats: 0 through 11
  Tones: 0 through 9
  Flags (set/clear): 0 through 29
  Flags (test): 0 through 55

- In addition to normal HP-41 number entry lines, certain forms carried over from the HP-67/97 are acceptable:

Table 3-1. Additional Numeric Entry Lines

HP-67/97 Form	HP-41 Emulator Equivalent	Examples
.	0	.
E	1	E
Emmm	1Emmm	E1, E234
E+mmm	1E+mmm	E+4, E+286
E−mmm	1E−mmm	E−2, E−456

- You can include comments in your file by enclosing the comments in parentheses. A comment can follow the HP-41 function, or it can be placed on a separate text line.

  Example: The following program illustrates entering an HP-41 program, with comments, into the editor.

```
(Count to five program)
LBL "CT5"
1.005 (Loop control number)
FIX 0 (Show only one digit)
LBL 1 (Start of loop)
VIEW X (Display current number)
PSE (Pause for one second)
ISG X (Increment x)
GTO 1 (Loop if x<6)
```

- You do not need an END function at the end of your user language program file written on the editor. If you do include one, (programs transferred from the HP-41 will always have a final END), it must be the last line of the file.

Using TRANS41

TRANS41 is a BASIC language program that creates a translated *intermediate* file from a text file containing an HP-41 user language program (*program-text* file). The intermediate file is in a form suitable for loading into the HP-41 environment. The translation is entirely automatic; you need only specify the names of the program-text file and the intermediate file.

The instructions below describe each step of the translation process. Examples use the program-text file THREE created in the previous section.

1. From the BASIC environment, type:

   ```
   RUN TRANS41
   ```

   The PRGM annunciator turns on, and the HP-71 prompts you for a file name:

   ```
   HP-41 Program File?
   ```

2. Type the name of the HP-41 program-text file to be translated. The file name can be any legal HP-71 file name, and can include extensions for a port number or a mass storage device. When you have typed in the name, press [END LINE].

   Example: Type THREE [END LINE].

3. In response to the prompt:

   ```
   Intermediate File?
   ```

   type the name of the intermediate file that will contain the translated program. This file name can have the extensions :MAIN or :PORT, but mass storage files are not allowed. (If you specify a mass storage file, the program displays RAM only and prompts you again). Press [END LINE] to terminate the file name entry.

   Example: Type THREEI [END LINE]

   TRANS41 creates the intermediate file by first duplicating the program-text file. (If the program-text file is stored on a mass storage device, it is copied into RAM automatically.) If you specify the same name (including extensions) for the intermediate file as for the program-text file, the program-text file is overwritten (with no warning). If the file name already exists but is not the name of the program-text file, the system beeps and displays:

   ```
   File exists. Purge?
   ```

   Pressing [Y] purges the existing file. To save the existing file, press any other key; the system prompts for another file name.

   When the intermediate file has been created, TRANS41 begins the translation. During the translation, the system displays:

   ```
   Translating... nnn
   ```

   where *nnn* is the line number in the program-text file currently being translated. The translation is performed from the last line to the first. In the example, *nnn* starts at 8 and counts down to zero.

   If you specified the name of a previously translated intermediate file in step 2, TRANS41 recognizes that the file does not need to be translated, and the HP-71 displays:

   ```
   filename already translated
   ```

   for one second, followed by:

   ```
   Load?
   ```

   Respond to this prompt as described in step 4.

4. When the translation is complete, the HP-71 sounds a tone and displays:

   ```
   mmm bytes. Load?
   ```

   where *mmm* is the number of bytes of HP-71 memory required to load the translated program into the HP-41 environment (86 bytes in the example). Load? asks you whether you want to load the program into the emulator or to exit TRANS41. Pressing any key except [Y] ends execution of TRANS41 and displays Done. You can then save the intermediate file on mass storage or magnetic cards for use later.

5. If you press [Y] in step #4, the translated program is loaded into emulator memory. (If the HP-41 environment has not been initialized, you will be prompted for data register memory size, as described in Section 2.) When the actual loading begins, the system displays:

   ```
   Loading... alpha labels
   ```

As the program is loaded, any alpha labels present in the original program-text file are displayed. For example, loading the example program displays:

```
Loading...THREE
```

If the program contains no global label, it receives a global label identical to the name of the intermediate file.

When loading is complete, the HP-71 sounds the familiar HP-41 4-tone beep, and displays `Done.`

6. If you enter the HP-41 environment after loading a program, the program pointer is positioned at the beginning of the newly loaded program. If the program requires number entry, use the emulator to enter the data. (Appendix F shows several examples of number entry.) The program can then be run by typing:

   `RUN` [END LINE]

Example: If you have followed along with the example program `THREE`, type:

`HP41` [END LINE]
`RUN` [END LINE]

The program `THREE` is executed, displaying:

```
RESULT=3.00
```

Halting Translation

During the translation countdown (the system is displaying `Translating...` *nnn*), translation can be stopped by pressing [ATTN]. The system displays:

```
Translating... Halt?
```

If you respond to the prompt by pressing any key but [Y], translation resumes. If you press [Y], translation halts. The partially-translated intermediate file remains in memory. If, in step 2 of translation, you specified the same name for the intermediate file as for the program-text file, you can restore the file to its original contents using the editor. If you have another copy of the file, however, it is easier to purge the partially translated file.

TRANS41 Errors

If any errors occur during execution of `TRANS41`, the HP-71 beeps and displays an appropriate BASIC or FORTH error message. For example:

- If there is not enough memory present to perform the loading operation, the system displays:

  ```
  Insufficient memory
  ```

- If a line contains too many characters, the system displays:

  ```
  String Ovfl
  ```

  with the relevant line number.

- If your program contains a function not included in the HP-41 emulator, or if you've misspelled a function, the system displays:

  not recognized

  with a program-text file line number.

Occasionally, you may see the warning message during loading:

@file name not unique*

The warning indicates that the name of the intermediate file being loaded is identical to the name of a previously loaded intermediate file. The message is a reminder that you may have two copies of the same program in emulator memory.

If an error message is too long to fit in the display, it can be scrolled right and left using the cursor keys.

Loading Intermediate Programs In the HP-41 Environment

An intermediate (translated) program can be loaded into emulator memory from the HP-41 environment using the LOAD function. The syntax for LOAD is:

LOAD *file name*

where *file name* is the name of an intermediate file. The *file name* can include a mass storage extension; you can load a program directly from mass storage without first copying the intermediate file into HP-71 memory.

The LOAD function alters the contents of the stack registers if the program being loaded contains number entry program lines.

The same errors can occur during execution of LOAD as during the loading portion of TRANS41. In addition, the message Bad Intermediate File can occur when attempting to load a file not created by TRANS41 if:

- The file does not exist or is not an HP-71 TEXT-type file.
- The file is not a proper intermediate file. (After LOAD determines that the file is a text file, it treats the file as an ordinary FORTH source code file and attempts to execute each line in the file as if it were typed in at the keyboard.)

* The message not unique is a standard FORTH warning message. Each HP-41 program compiled into emulator memory is entered into the FORTH dictionary as a word named *@file name*.

Halting Program Loading

To halt loading of a progrm by TRANS41 or LOAD, press [ATTN]*. (The [ATTN] key is disabled at certain times during loading, so you may have to press the key more than once.) When the loading operation has been halted, the HP-71 beeps and displays:

```
Halted
```

The partially loaded program is cleared from emulator memory.

Transferring Programs from an HP-41 using READ41

The program READ41, which is executed from the BASIC environment, transfers programs directly from HP-41 memory to an HP-71 text file. Use of READ41 requires an HP 82160A HP-IL Module for the HP-41, and an HP 82401A HP-IL module for the HP-71. Refer to owner's manuals for the two modules, if necessary, for additional information.

1. To connect the HP-41 and the HP-71, install the HP-IL modules into the appropriate plug-in ports. Then plug the cables attached to the HP-41 HP-IL module into the sockets of the HP-71 HP-IL module. You can attach other HP-IL devices on the loop, but they must be controlled by the HP-41.
2. Place the HP-41 in manual I/O mode, using the MANIO function:

   [XEQ] [ALPHA] MANIO [ALPHA]
3. Make the HP-71 the selected device:

   n [XEQ] [ALPHA] SELECT [ALPHA]

   where *n* is the HP-IL address of the HP-71, relative to the HP-41. If the HP-71 is the only device connected to the HP-41, then n = 1.
4. Clear HP-41 flags 15 and 16 (CF 15 and CF 16) to place the HP-41 in manual printer mode. (The HP-41 treats the HP-71 as if it were a printer device.)
5. Ready the HP-71 by typing:

   ```
   RUN READ41 [ENDLINE]
   ```
6. In response to the prompt:

   ```
   HP-41 Program File?
   ```

   respond by typing in the name of the program-text file that will contain the program transferred from the HP-41. If a file by that name already exists, the HP-71 beeps and display:

   ```
   File exists. Purge?
   ```

   Pressing [Y] purges the existing file. To save the existing file, press any other key. The program will prompt you for a new file name. When you have entered the file name, the system displays:

   ```
   Reading HP-41...
   ```

   to indicate that the HP-71 is now awaiting a transmission from the HP-41.

* INIT 1 and INIT 2 also halt loading, but do not leave emulator memory intact. If you have halted loading using INIT 1 or INIT 2, you must reinitialize the emulator with purge41.

7. "Print" the HP-41 Program to the HP-71. On the HP-41, press:

   [XEQ] [ALPHA] PRP [ALPHA] [ALPHA] *label name* [ALPHA]

   where *label name* is the name of any global alpha label in the HP-41 program. If you omit the *label name*, the HP-41 sends the program currently containing the program pointer.

   As the transfer takes place, the HP-71 displays each HP-41 program line as it is received from the HP-41. If you are using an HP-41CX, or an HP-41C or CV with a HP 82182A Time Module, the time and date are displayed on the HP-41.

8. When the program transfer from the HP-41 to the HP-71 is complete, the HP-71 beeps and prompts:

   ```
   Translate?
   ```

   If you press any key other than [Y], the HP-71 displays `Done` and execution of `READ41` terminates. If you press [Y], `READ41` automatically runs `TRANS41`, entering that program at the point where the `Intermediate File?` prompt is displayed. Continue at step 3 of the `TRANS41` instructions on page 42.

READ41 Errors

Three types of errors can occur during a program transfer by `READ41`:

- If a transfer halts due to a transmission interruption between the HP-41 and the HP-71, the line at which transmission halted remains in the HP-71 display. Then, within a minute, the HP-41 displays:

  ```
  Transmit error
  ```

  The interruption can be accidental, such as a power failure in one of the HP-IL devices or disconnection of one of the loop connectors. You can cause a deliberate interruption, by pressing the [ATTN] key on the HP-71, or the [ON] key on the HP-41.

- The message:

  ```
  Bad HP-41 print mode
  ```

  is caused by an incorrect printing mode on the HP-41. If you have forgotten to clear HP-41 flags 15 and 16, the program is transferred in a format impossible for `TRANS41` to translate correctly. When `READ41` detects that the transferred program is not in the required format, it stops the transmission, beeps, generates the error, and purges the file containing the partially transferred program.

- The HP-71 beeps and displays the message:

  ```
  Insufficient Memory
  ```

  if it runs out of memory during the file transfer. The partially transferred program file is not purged, since its contents may still be useful to you.

All three types of errors can leave the HP-IL loop in a indeterminate state, so that you will need to carry out a few recovery steps to insure that a retransmission of the program will proceed properly:

1. Wait until the HP-41 displays `Transmit error`. Then, turn the HP-41 off.
2. If the HP-71 PRGM annunciator is still on, press the [ATTN] key twice. The HP-71 will display `HPIL ERR: Aborted`.
3. Type `RESET HPIL` [ENDLINE] on the HP-71.

4. Turn the HP-41 back on, and correct the error that caused the interruption:
   - If the error was a bad HP-41 print mode, clear flags 15 and 16 on the HP-41.
   - If the HP-41 displayed TRANSMIT ERROR and you did not press [ATTN] on the HP-71, there was a loop failure. Check the HP-IL cables and the battery levels or AC power connections on all the devices connected on the loop.
   - If the error was Insufficient Memory, you will have to purge one or more files from HP-71 memory to make room for the incoming file.
5. Start the program transfer again.

HP-71 Memory Usage

An HP-41 program translated and compiled into the HP-41 environment requires, on the average, approximately 2.5 times as much memory in the HP-71 as its original form did in the HP-41. Although the HP-71 memory is large enough to compensate for this increase in program size, the use of program-text files and intermediate files by TRANS41 makes it possible to have three separate versions of the same program in HP-71 memory. This uses up memory rapidly.

If you have an HP-71 card reader, you can change procedures so that no more than two of these three program forms are in HP-71 memory at the same time. If you have an HP-IL module and an HP-IL mass storage device, only one form of a program needs to be in memory at any stage of the translation. Here are some tips for occasions when memory space is limited:

- **For the HP-71 Card Reader:**
  1. Copy the FTH41RAM file to cards, then purge it.
  2. Create your program file, and copy it to cards.
  3. When you run TRANS41, use the same name for the intermediate file as for the program file.
  4. Copy FTH41RAM back into memory, and use LOAD in the HP-41 environment to compile the intermediate file into emulator memory.
- **For HP-IL mass storage:**
  1. Copy the FTH41RAM file to mass storage, then purge it.
  2. Create your program file, copy it to mass storage, and purge it from HP-71 memory.
  3. When you run TRANS41, specify the program file with the appropriate mass storage file specifier.
  4. Copy the intermediate file to mass storage, without loading it.
  5. Reload the FTH41RAM file, and use LOAD in the HP-41 environment to load the intermediate file directly from mass storage to emulator memory.

Creating New Functions

One of the major strengths of the FORTH language is that it can be extended. A programmer can easily add new functions (called *words* in FORTH) to the language. In fact, there is no distinction between programs and functions; the entire process of programming in FORTH consists of adding functions. The HP-41 emulator, which is based on a FORTH language system, can also be extended. You can use FORTH to add HP-41 functions that are missing from the emulator, or to create new functions of your own.

Suppose, for example, that you are writing a program that recalls the system clock time and converts it to seconds. On the HP-41, you would write a subroutine to perform the calculation:

```
LBL 01
TIME
HR
3600
*
RTN
```

On the HP-71, you can write the same subroutine, using the methods described previously in this section. Or, you can use the FORTH system to write a new function to perform the entire calculation, which you can then use in any program that you subsequently load into emulator memory.

The process for entering a new FORTH function into the emulator is:

1. Execute `HP41`, if necessary, to enter the HP-41 environment.
2. Expand the `FTH41RAM` file to make room for the new function by typing:

   number of nibbles `XSIZE` [END LINE]

   to make memory available for your new function (Two nibbles equal one byte.) You can add as many nibbles as you need, within the limits of available memory. The formula for computing the number of nibbles required by a function is:

   $20 + 2 \times (1 +$ *name length*$) + 5 \times ($*number of functions in the definition*$)$

   where *name length* refers to the name of the new function. Usually it's easier to expand the file by a large amount than it is to try to determine the exact requirements in advance. After adding functions, you can reclaim any unused space (step 4).
3. Type in the new function in the format:

   `:` *NAME function1 function2 function3 ... functionN* `;`

   where *NAME* is the name you choose for the new function, and *function1*, *function2*, etc., are the names of existing HP-41 (or FORTH) functions, separated by spaces.

   Example: A sample new function, which we might call `^TIME`, has the following definition:

   `: ^TIME TIME HR 3600. * ;` [END LINE]

   The definition does not have to fit on a single line. You can press [END LINE] after entering any part of the definition; the definition does not end until you type in the final `;`.

   You can make as many definitions as you like, depending on available memory. If you run out of space in the `FTH41RAM` file, you will see the error message `FTH ERR: dictionary full`. When that happens, you must execute `XSIZE` to expand `FTH41RAM`. If there is not enough room in HP-71 memory to expand the `FTH41RAM` file as much as you specify, `XSIZE` returns the error message `Insufficient memory`.

   There are some restrictions that apply to adding new HP-41 functions using FORTH:

   - Function names can be 1 through 31 characters long, and cannot contain spaces, quotes (`"`), or question marks (`?`).
   - Names should not match any existing HP-41 function names or FORTH words.
   - Do not start new function names with `E+`, `E-`, or `E` followed by a number digit, or with `ISG` or `DSE`.

- Only one-part (one byte on the HP-41) functions can be used in the definition of new functions. You can't use register functions, flags, tones, GTO, XEQ, etc., nor any function containing text.
- When the first function in a definition raises the (floating-point) stack (for example, TIME, ATOX, LASTX), you must start the new function name with the character ^. Do not use ^ as the first character otherwise. A leading ^ is used as a signal to the translator program so that it knows how to handle stack lift disable situations.

4. After adding functions, you can reclaim any unused space by typing:

   0 XSIZE [END LINE].

Section 4

The Editor

The HP-41 Pac enables you to create, modify, copy, list, and print text files. These files are suitable source files for the FORTH system and the HP-41 emulator. This section describes the editor's operation in three parts:

- "Overview of the Editor" describes how to enter and exit the editor, the two types of editor commands, and editor operations other than commands.
- "Editor Commands" describes the specific commands that act on the edit file.
- "Editor Files" describes files used in the editor's operation.

Additional material related to the editor appears in the appendixes. Appendix B, "Error Messages," includes the error messages generated by the editor. Appendix C, "BASIC Keywords," includes the editor keywords DELETE#, EDTEXT, FILESZR, INSERT#, MSG$, REPLACE#, SCROLL, and SEARCH, which you can use in your own BASIC programs.

Overview of the Editor

The editor is a BASIC program; when you enter the editor, the HP-71 PRGM annunciator appears. You can enter the editor directly from the FORTH or HP-41 environments by using BASICX. Typing:

```
" EDTEXT SCREEN" BASICX [END LINE]
```

runs the editor on a file named SCREEN. When you exit the editor, the HP-71 automatically returns to the original environment.

To enter the editor from BASIC, type EDTEXT *file name* [END LINE]. The editor opens that file for editing or, if *file name* is a new name, creates a new file with that name. The display then shows Line *n*, Cmd:, where line *n* is the current line in the file. Line numbers, which begin with 1, are for reference only; they aren't stored in the file. If you have created a new file, or if you're at the end of the file, the current line is indicated by Eof.

When the Cmd: prompt is displayed, you can:

Display the Current Line. To temporarily display the current line, hold down the [END LINE] key. When you release the key, the Cmd: prompt returns.

Move to A Different Line. There are three methods for moving to a different line:

- To move to any line in a file, enter the line number and press [END LINE]. For example, to move to line 2, enter 2 [END LINE].
- To move to the previous line (smaller line number), press [↑]. To move to the following line (larger line number), press [↓].
- To move to the beginning of a file, press [g][↑]. To move to the end of a file, press [g][↓].

Display the File Name. If you press [↑] when the line 1 is the current line, the editor displays the name of the edit file. To display the file name from any place in the file, hold down [f][+]. When you release [+], the Cmd: prompt returns.

Execute a Command. The editor commands, each of which is described in detail below, fall into two classes:

- The commands T (Text) and I (Insert) are used for entering text. Once you execute the Text or Insert command, the editor remains in Text or Insert mode until you press [RUN] or [ATTN]; only then will the Cmd: prompt return.
- All other editor commands perform specific operations, after which the Cmd: prompt returns automatically.

Exit the Editor. To end the editing session, enter E [END LINE]. The editor closes the edit file and displays Done: *file name*. If you decide not to keep this file, purge it following the instructions in section 6 of the *HP-71 Owner's Manual.*

When you call the editor, a copy of your own redefined keyboard is stored and the editor's key redefinitions are added to yours. Unless the editor keys are the same keys you've redefined, your redefined keys are still available to you while the editor is active. When you exit the editor, the combined redefined keyboard is purged and your own redefined keyboard is restored.

To override a key assignment, use the [g][1 USER] key. This will deactivate USER mode for the next key pressed. Note that if you enter the editor from the FORTH or HP-41 environments, disable USER mode, and then either press [ATTN] or cause any error, the HP-71 immediately returns to the FORTH or HP-41 environment, leaving the current edit file in a corrupted state.

Editor Commands

You can enter the following editor commands whenever the Cmd: prompt is displayed. Some editor commands require parameters such as line numbers or a file name. These parameters are identified in syntax diagrams for each command. Any default values for parameters are given after the syntax diagram. In the syntax diagrams:

- Items [enclosed in square brackets] are *optional parameters*. Some optional parameters are nested within others. This indicates that the parameter in the outer pair of brackets must be present *before* the parameter in the inner pair can be included.
- Items shown in DOT MATRIX text must appear exactly as shown (although either upper or lower case is acceptable).
- There are two substitute characters that can be used for any *line-number* parameter. A period (.) indicates the *current line*, and the pound sign (#) indicates the *last line in the file*.
- Two adjacent numeric parameters must be separated by a space or comma. No separation is required between a numeric parameter and an alphabetic parameter.

The Text (T) and Insert (I) Commands

[*line number*] T

[*line number*] I

Default value: *line number* = current line

The Text command is your primary means of adding text to the edit file. When you enter Text mode, the current line appears in the display with the cursor at the beginning of the line. Modify the current line as desired (using the standard HP-71 editing keys) and then press [END LINE]. The editor stores these changes to the current line and then makes the following line the current line, displaying it to start the cycle again.

The Insert command permits you to add a line or a series of lines into the middle of a file. When you enter Insert mode, the current line is displayed until you press a key. Type in the text for the new line (using the standard HP-71 editing keys) and press [END LINE]. The editor inserts the new line into the file, just before the current line, and then displays the next line number as the new current line. (The text for the new current line is the same as before; only its line number changes.) Flag one is on to indicate that you are in Insert mode.

Either Text mode or Insert mode work equally well for entering text at the end of a file. In either mode, text is stored in the file only when you press [END LINE]. If you make changes or enter text and then move to another line (by using [↓] or [↑]) *before* you press [END LINE], no changes or text will be stored.

To exit from Text or Insert mode, press [RUN] or [ATTN].

The List (L) and Print (P) Commands

[*beginning line number* [*ending line number*]] L [*number of lines*][N]

[*beginning line number* [*ending line number*]] P [*number of lines*][N]

Default values: *beginning line number* = current line
ending line number = last line

The List and Print commands are similar. List causes the specified lines of text to be displayed consecutively on the current display device (usually the display window or ▪ monitor). If you have an HP 82401A HP-IL Interface installed and a printer assigned, Print causes the specified lines to be printed. When no printer is present, Print responds like List.

After listing or printing, the *current line* will be the line after the ending line number. The following examples show some List and Print commands with parameters:

L	List from the current line to the end of the file.
.L10	List from the current line to the end of the file, or just 10 lines, whichever comes first.
3 9 L N	List from line 3 to line 9 with line numbers.
1 L20N	List, with line numbers, the entire file or the first 20 lines, whichever comes first.
P	Print from the current line to the end of the file.
.P5N	Print five lines starting at the current line, with line numbers.
1P N	Print the entire file with line numbers.

The Copy (C) and Move (M) Commands

[*beginning line number* [*ending line number*]] C [*filename*]

[*beginning line number [ending line number*]] M [*filename*]

Default values (Edit file): *beginning line number* = current line
ending line number = beginning line number

(Other file): *beginning line number* = line 1
ending line number = last line

The Copy command permits you to copy one or more lines from one place in the file to another place in the file. You can also copy part of another file into your edit file. Copy always inserts the copied text before the current line. The Move command is similar to the Copy command but deletes the text in the original location.

If no filename is specified, the indicated lines come from the edit file. If a filename is specified, the indicated lines come from the specified file. You can't copy or move a block of text that includes the current line, unless the current line is the first or last line of the block of text.

The `Working...` message is displayed when you copy or move text.

Here are some examples of the Copy and Move commands:

`C`	Duplicate the current line.
`5 C`	Copy line 5 and insert it before the current line.
`3 9 M`	Move lines 3 through 9 from within the edit file and insert them before the current line, then delete the original lines 3 through 9.
`C CAT`	Copy the file `CAT` and insert the lines before the current line.
`20 C ABC`	Copy lines 20 through the last line of the file ABC and insert the lines before the current line in the edit file.

The Delete (D) Command

[*beginning line number* [*ending line number*]] `D` [*filename* [+]]

Default values: *beginning line number* = current line
ending line number = beginning line number

The Delete command deletes one or more lines from the edit file. You can place the deleted lines into a new file or, using the + option, append the lines to an existing file. When you execute Delete with line number parameters specifying more than one line, the message `OK to delete? Y/N:` will appear. You must answer `Y` before the editor will complete the deletion. If you answer `N`, the Command Prompt returns.

The `Working...` message is displayed when you use Delete.

The following examples show some uses of the Delete command:

`D`	Delete the current line.
`12 32D`	Delete lines 12 through 32.
`4 9 D CACHE`	Delete lines 4 through 9 and store them in a new file called CACHE.
`2 21D ARCHV+`	Delete lines 2 through 21 and append them to the end of a file called ARCHV.

You can not purge a file while you are in the editor, but you can delete all of the text and leave an empty file. Refer to section 6 of the *HP-71 Owner's Manual* for instructions on how to purge a file.

The Search (S) and Replace (R) Commands

[*beginning line number* [ending line number]][?] S/*string1*[/]

Default values: *beginning line number* = current line + 1
ending line number = last line

[*beginning line number* [*ending line number*]][?] R/*string1*/*string2*[/]

Default values: *beginning line number* = current line
ending line number = beginning line

The Search and Replace commands allow you to search through a file for a certain string of characters *string1*. If you use a Search command, the first line containing *string1* becomes the current line. If you use a Replace command, all occurrences of *string1* are replaced by *string2*, and the last line containing *string1* becomes the current line. If either command can't find *string1*, it displays `Not Found`.

These commands search the specified lines in the edit file for the string indicated between the slashes (/). These slashes act as *delimiters*, marking the string's boundaries. If you need / as a normal character in your search string, you can use any other character (except a blank space) as the delimiter. The first non-blank character after the command S or R is the delimiter. The last delimiter is optional unless another command follows this command.

Search and Replace can distinguish between uppercase and lowercase letters. For example, a search for the string `jack` will not find the string `Jack`.

The following examples show some Search commands and Replace commands with parameters:

`S/Jack`	From the next line through the end of the file, search for the first occurrence of the string "Jack."
`3 7 S/Jill`	From line 3 through line 7, search for the string "Jill."
`R/cat/dog/`	Replace all occurences of "cat" with "dog" on the current line.
`4 7R/cat/dog`	On lines 4 through 7, replace all occurences of "cat" with "dog."
`R*3/4*3/8`	On the current line, replace all occurences of "3/4" with "3/8." The character * is used as the delimiter so that slashes may occur in the strings.
`.#R/meet//`	From the current line to the end of the file, replace "meet" with the null string (that is, delete "meet").

If the replacement *string2* causes the line to be longer than 96 characters, the editor will redimension variables, causing a slight delay.

Response Option. You can more closely control the Search and Replace commands by including the ? option in the command string. With this option the editor stops with each match to *string1* and waits for you to respond. The display shows the following information:

- The number of the line containing the matching string.
- The number of the column in which the first letter of the matching string occurs.
- A backslash (\) delimiter.
- Some of the line, beginning with the matching string.
- A slash (/) delimiter.
- A question mark (?) indicating that a response is expected.

Responding to a Search command, your options are:

- Press [Y] to stop the search at this match and make this line the current line.
- Press [N] to search for the next occurrence of the string.
- Press [Q] to quit the search and return to the previous current line.

Responding to a Replace command, your options are:

- Press [Y] to replace this occurence of *string1* with *string2* and search for the next occurrence of *string1*.
- Press [N] to leave this occurence of *string1* intact and search for the next occurrence of *string1*.
- Press [Q] to quit the replacement search and make the last line where replacement occured the current line (or return to the previous current line if no replacements occurred).

If you press any other key (except [ATTN]), the display will show `Y/N/Q ?` to indicate that only Y, N or Q are permitted as responses. If you press [ATTN], the `Cmd:` prompt returns.

The Replace command can result in lines longer than 96 characters. If this occurs while you're using the ? option, you can scroll through only a 96-character substring that contains that search string, not through the whole line.

Defining Patterns in Strings. Five characters (., @, &, ^, and $) can have special meanings when you're defining strings. To switch these characters to their special meanings, place a backslash (\, assigned to [f][/]) in the string; to return these characters to their normal meanings, place a second backslash in the string. (The string's final delimiter also returns the characters to their normal meanings.) Any of these five characters appearing between the two backslashes will be given their special meaning.

The five characters, their special meanings and some examples of their uses are described in the following paragraphs:

- The period (.) represents any character, and so is called a *wild-card* character. When the editor searches for a matching string, any character can be in those positions where you put a period.

  Example. `R/ABC\...\/Recheck ID#/` will replace the occurrences of ABC followed by any three characters, such as `ABC999`, `ABCzyz`, or `ABC yz`, with the string `Recheck ID#`. `R/ABC\...\/Recheck ID#` has the same effect; the second backslash is not needed because the end of *string1* stops the special-meaning feature, and the ending slash is optional for *string2*.

- The commercial "at" symbol (@) represents any number of wild-card characters. Because the program starts searching for the end of the string at the end of the line, the longest match possible is found.

  Example. R/ABC\@\CDE/Recheck ID#/ will replace any string that begins with ABC and ends with CDE, such as ABC123CDE, ABCCDE, or ABC12 zzzCDE, with the string Recheck ID#.

- The ampersand (&) represents the text that matches *string1*; it is used in a Replace command to insert the actual string that matched *string1* (which may include wild cards) into *string2*.

  Example. R/\AB.*/\&DEF/ searches for the string AB*wildcard* and appends the string DEF to it. If ABC is found, the new string will be ABCDEF.

- The up-arrow (^) represents the beginning of a line. As the first character in a string, it specifies that a matching string must be at the beginning of a line. If the up-arrow isn't the first character in the string, it has its normal meaning.

  Example. R/\^\ABC/CDE/ will search for the string ABC only at the beginning of a line. If ABC appears anywhere else in the line, a match will not be made.

  Example. Suppose you have loaded a text file from the HP-75 into your HP-71. Now you want to delete the four-digit line numbers that the HP-75 put at the beginning of every line. 1#R/\^..../// tells your HP-71 to search, from line 1 to the end of the file, for any four characters at the beginning of the line, and replace them with nothing (delete them).

- The dollar sign ($) represents the end of a line. As the last character in a string, it specifies that a matching string must be at the end a line. If the dollar sign isn't the last character in the string, it has its normal meaning.

  Example. R/ABC\$/CDE will search for the string ABC only at the *end* of a line. If ABC appears anywhere else in the line, it will be ignored. A second backslash is not needed after the $ because the dollar sign is at the end of *string1*.

If you need to search for a string containing a backslash character as part of the text, you don't want Search and Replace to see the backslash as a switch. The solution is to use two sequential backslashes. The editor will interpret \\ as a single backslash character, not as a switch.

Editor Files

The editor uses several files in its operation. The names of these files must not be used as the names of files in the HP-71 user memory, because the HP-71 first searches its own memory before searching the plug-in modules. The following list gives the name of each file in the module, along with a brief description of the file.

EDTEXT	The editor BASIC language program.
EDLEX	A LEX file containing the assembly level support for the editor, including the BASIC keywords.
EDKEYS	The editor keys file.
EDUKEYS	A temporary keys file created by the editor in main memory to store your user defined keys while the editor is running. When you exit the editor, these keys again become current.

Section 5

The HP-71 FORTH System

Introduction

The HP-41 Pac contains a FORTH system tailored to the HP-71. The advantages of FORTH over BASIC are speed and complete access to the machine. Programs can be written in FORTH, in BASIC, or in both, making use of the best features of each language/system.

FORTH *secondaries* (words constructed from existing FORTH words) can be compiled from keyboard input or from text files created by the editor. The editor is discussed in section 4.

The word set of the HP-71 FORTH kernel is similar to that defined in the FORTH-83 Standard. This section describes their differences in "Unique Aspects of HP-71 FORTH," which covers enhancements and methods of implementation that are machine-related, and in "FORTH Extensions," which covers enhancements not directly tied to the HP-71. For the complete definition of any FORTH word, standard or nonstandard, refer to appendix D.

References

This section doesn't contain the complete FORTH-83 Standard or tutorial information about FORTH; you can find such material in the following books. You will need to keep in mind the unique aspects of HP-71 FORTH as you read these books.

- Brodie, Leo. *Starting FORTH.* Englewood Cliffs, N.J.: Prentice-Hall, 1981. An effective and entertaining introduction to FORTH.
- *FORTH-83 Standard.* Mountain View, Ca.: FORTH Standards Team, 1983.
- Haydon, Glen B. *All About FORTH: An Annotated FORTH Glossary.* Second edition. Mountain View, Ca.: Mountain View Press, 1983. Some definitions in this manual are borrowed from Dr. Haydon's book.

Using FORTH on the HP-71

Entering and Exiting FORTH. To enter the FORTH environment, type the BASIC keyword `FORTH` and press [END LINE]. The computer displays the FORTH sign-on message `HP-71 FORTH` and the version. To exit the FORTH environment, type the FORTH word `BASIC` or `HP41` and press [END LINE].

The RAM-based portion of the FORTH system, including user-added dictionary words, is contained in an HP-71 file named FTH41RAM. When you exit FORTH, either by executing BASIC or by pressing the [OFF] key, the contents of the FTH41RAM file are preserved. Thus the FORTH environment will be in the same state when you reenter as when you exited. If you turn off the HP-71 from the FORTH environment, it returns directly to the FORTH environment when you turn it on. If you purge the FTH41RAM file from the BASIC environment, a new FTH41RAM file will be created when you next execute FORTH.

User Prompts. If you press [END LINE] while the HP-71 FORTH prompt is displayed, FORTH displays OK < 0 >. The OK indicates that FORTH is ready to accept input, and the 0 indicates how many items are on the data stack. If you then type 1 2 3 [END LINE], the FORTH system displays OK < 3 >. You can suppress the OK message by storing a non-zero value into the user variable OKFLG.

Line-editing Keys. All of the HP-71 line-editing keys are functional while in the FORTH environment. Pressing [ATTN] while entering a line clears the display and leaves only the blinking cursor.

Key Redefinitions. The FORTH system duplicates the BASIC method of handling redefined keys. You can switch in and out of user mode while in FORTH, but you must be in the BASIC environment (or use BASICK) to redefine keys.

The Command Stack. The HP-71 command stack is available in FORTH. It operates just as in BASIC, except that in FORTH you can enter the Command Stack by pressing any of the up- or down-arrow keys—you don't need to press [9][END LINE] first.

Exceptions and the [ATTN] Key. Because the FORTH system can run a program for an indefinite time, it must occasionally check whether a system exception has occurred. FORTH checks for exceptions when it executes ; (semicolon) in a secondary and before it branches in a loop structure. If an exception has occured, FORTH issues the exception poll. An exception can be a service request from the HP-71's internal timers or from other devices, or can result from pressing the [ATTN] key.

Pressing [ATTN] stops the execution of any FORTH word (except HP-IL words, which require pressing [ATTN] twice). Once the FORTH environment recognizes that [ATTN] has been pressed, it executes the system equivalent of ABORT to reset the data and return stacks and to restart the FORTH outer loop (the FORTH system user interface).

Errors. If an error occurs in the FORTH system, all files are closed and an error message is displayed. FORTH error messages sound a tone and preface all errors with FTH ERR:. FORTH error numbers and messages are available through the BASIC keywords ERRN and ERRM$.

If an error occurs in a BASIC O/S subroutine called by the FORTH system, the error message appears as ERR: rather than FTH ERR:.

Unique Aspects of HP-71 FORTH

Twenty-Bit FORTH

Most FORTH systems are implemented on byte-oriented machines with 16-bit addresses. The HP-71, in contrast, is a nibble-oriented machine with 20-bit addresses. To allow access to the entire 1M-nibble address space and to achieve maximum speed, FORTH on the HP-71 is a 20-bit implementation. That is, the data and return stacks are 20 bits wide, and the addresses on those stacks are 20-bit absolute addresses. All quantities on the stacks are 20-bit quantities, regardless of whether a one-byte or 20-bit operation is performed. Unused high-order nibbles are zero or are expected to be zero.

HP-71 FORTH conforms to the FORTH-83 Standard in intent but, because of the nature of the HP-71 CPU, not exactly in effect. The functionality of the Standard required word set, plus selected words from the extension word sets, are provided in HP-71 FORTH. In most cases, the HP-71 uses the same word names as the Standard. You can determine the behavior of particular HP-71 words compared with their Standard counterparts according to the following general guidelines:

- For operations that deal with *bytes* (such as C@, CMOVE, and FILL), the Standard names are retained for HP-71 FORTH words. Such words will produce the same result as the corresponding Standard words. In several cases analogous words that deal with nibble quantities are also provided; they are listed below in "Nibble and Byte Words."
- For operations that deal with *cells* (such as +, @, and CONSTANT), the Standard names are retained for HP-71 FORTH words. Such words will produce the same result as the corresponding Standard words, except that the quantities manipulated by the words are 20 bits long instead of 16.
- For operations that don't translate well to the HP-71 (with its continuous memory and multiple-file system), the Standard names are replaced by HP-71 FORTH words. For example, LOAD (load from a numbered screen) is replaced by LOADF (load from a named text file), and EXPECT (read up to a specified number of characters) is replaced by EXPECT96 (read up to 96 characters).

The table below lists those words that HP-71 FORTH adds to the Standard word set to perform nibble operations, together with their byte-oriented counterparts.

Table 5-1. Nibble and Byte Words

Nibble Word	Action	Byte Word	Action
NALLOT	Allot *n* nibbles.	ALLOT	Allot *n* bytes.
NFILL	Fill *n* nibbles.	FILL	Fill *n* bytes.
N@	Fetch one nibble.	C@	Fetch one byte.
N!	Store one nibble.	C!	Store one byte.
NMOVE	Move *n* nibbles.	CMOVE	Move *n* bytes.
NMOVE>	Move up *n* nibbles.	CMOVE>	Move up *n* bytes.
5+	Increment address by 5.	2+	Increment address by 2 (one byte).
5-	Decrement address by 5.	2-	Decrement address by 2 (one byte).

Compilation from Files

FORTH compiles new words into the dictionary from "screens" as well as from the keyboard. In traditional versions of FORTH, a screen is a 1K-byte block on a mass storage device (16 lines of 64 bytes each).

Screens. In HP-71 FORTH, a "screen" is a standard HP-71 text file. Each text file consists of a series of text strings of variable length, with each text string preceded by a two-byte length field. The file is terminated by a two-byte marker, FFFF. The editor, described in section 3, can create source screens for FORTH. The name of a screen must be a legal HP-71 file name. The maximum size line that FORTH will process is 96 bytes, which corresponds to the logical display size.

LOADF. The Standard word LOAD is replaced in HP-71 FORTH by LOADF. The inputs to LOADF are two 20-bit numbers: the length of the character string specifying the file to be loaded and the address of this string. LOADF calls HP-71 routines to open, read, and close the file. These routines, in turn, interface to the HP-IL module if it is present, so that screens can reside on HP-IL mass storage devices as well as in HP-71 memory.

FIB Entries. Executing LOADF opens the screen file and creates a *file information block* (FIB) entry in a system buffer called the FIB general purpose buffer. The FIB entry identifies the file and indicates whether the file is in RAM or on mass storage. (If the file is on mass storage, the FIB entry is linked to a system buffer called an I/O buffer that identifies the file.) A *file-information-block number* (FIB#) identifying the FIB entry is stored into the FORTH user variable SCRFIB (*screen FIB#*) to specify the active file.

Mass Memory Buffers. When a file is loaded, its FIB# and the first line of the file are read into a mass memory buffer. There are three mass memory buffers, used in rotation. The contents of the buffer are interpreted until the null at the end of the line (placed there by the FORTH system) is reached. The FORTH word WORD then determines whether this is the end of the active file and, if not, reads the next line from the file into the same mass memory buffer. Each mass memory buffer has the following format.

Format of a FORTH Mass Memory Buffer

FIB#	Line#	Byte count	Data	2 Nulls
1 byte	5 nibbles	2 bytes	Up to 96 bytes	2 bytes

LOADF can save the information necessary to return to the file it is currently interpreting, so LOADF commands can be nested.

Mass Memory. A user can LOADF a file from cassette or disk directly into the FORTH dictionary without first storing the file in RAM. The file will be interpreted a line at a time by reading the line into a FORTH mass memory buffer. However, a file stored on a magnetic card must be read into RAM before it can be loaded into the FORTH dictionary or edited.

File Words

- LOADF accepts input from a specified file rather than the keyboard. Words are executed and definitions are compiled into the user dictionary. The file may exist in RAM or on mass storage.
- BLOCK reads a specified line of the active file into a mass memory buffer and returns the address of the first data byte in the mass memory buffer.
- CLOSEF closes a specified file.
- EOF returns a true flag if the end of the active file has been reached, a false flag if not.
- +BUF returns the address of the next available buffer.
- OPENF opens a FIB entry for a specified file.
- CLOSEALL closes all open files.
- FIRST is a user variable containing the address of the first mass memory buffer in memory.
- LIMIT is a user variable containing the address of the first byte beyond the mass-memory-buffer area.
- PREV is a user variable containing the address of the mass memory buffer last used.
- USE is a user variable containing the address of the mass memory buffer to use next.
- SCRFIB is a user variable containing either the FIB# of the active file being interpreted by LOADF or else 0.
- BLK is a user variable containing either the line number of the file being interpreted by LOADF or else 0 (input from keyboard).
- LINE# is a user variable containing the line number being loaded from the file specified by SCRFIB.

FORTH/BASIC Interaction

The HP-41 Translator ROM enables you to temporarily enter the FORTH environment from within the BASIC environment, and vice versa, to take advantage of features of one system while operating from the other. If you press [ATTN] while in a temporary environment, you will be returned to the original environment.

BASIC to FORTH. There are four programmable BASIC keywords that access the FORTH environment.

- FORTHX is a BASIC statement, returning no result.
- FORTHF is a BASIC numeric function that returns the contents of the X-register in the FORTH floating-point stack.
- FORTHI is a BASIC numeric function that returns the number on the top of the FORTH data stack, dropping that value from the stack.
- FORTH$ is a BASIC string function that returns the string specified by the address and character count on the top of the FORTH data stack, dropping those two values from the stack.

FORTHF, FORTHI, and FORTH$ read data from the FORTH environment into BASIC variables without executing any portion of the FORTH system (although FORTHI and FORTH$ alter the data-stack pointer). FORTHX, however, enables you to transfer BASIC data to the FORTH environment and to execute any FORTH words before automatically returning to BASIC.

To execute FORTH operations from the BASIC environment, you use the keyword FORTHX followed by a command string plus up to 14 additional parameters. The optional parameters can be any combination of strings or numeric quantities. The numeric quantities will be pushed onto the FORTH data stack as single-length numbers; strings will be specified on the stack by their addresses and character counts. FORTHX first pushes the optional parameters onto the data stack and then executes the command string. The command string can contain any sequence of FORTH words and parameters, just like input you would enter from the keyboard.

Examples.

```
A$ = FORTH$
X2 = FORTHI
T = TAN (FORTHF)
FORTHX "DROP SWAP TYPE DEPTH .",10,20,X2,"Hello",2*X+6
```

For additional details, refer to appendix C, "BASIC Keywords."

FORTH to BASIC. There are four FORTH words that pass a string (specified on the data stack) to the BASIC system for execution. The string contains BASIC keywords and parameters. The FORTH words call the appropriate BASIC routines to parse and execute the string, as if it were typed to BASIC from the keyboard.

- BASICX passes a string containing BASIC statements to the BASIC system for parsing and execution. It returns no value to the FORTH environment. BASICX can alter the value of BASIC variables. If the string begins with a line number, it will be added to the current BASIC edit file. The string can also call BASIC programs. When the BASIC interpreter finishes, it issues a poll that allows the FORTH system to regain control. If an error occurs, the BASIC system reports the error to the user, and FORTH runs the system equivalent of the ABORT word.
- BASICF passes a string containing a numeric expression to the BASIC system for evaluation. It returns the value of the numeric expression to the X-register in the FORTH floating-point stack.
- BASICI passes a string containing a numeric expression to the BASIC system for evaluation. It returns the value of the numeric expression to the FORTH data stack.
- BASIC$ passes a string containing a string expression to the BASIC system for evaluation. It returns the resulting string to the PAD area and the address and character count to the data stack. The resulting string is truncated to 255 characters if it exceeds this length.

Examples.

```
" BEEP" BASICX
" A(1)" BASICI
" A5/B4*PI" BASICF
" A$" BASIC$
" A6=T(4)*PI" BASICX
" 50 DISP A,B;" BASICX
" A4" BASICF
" STATUS" BASICI
" TIME" BASICF
```

The FORTH/BASIC interface is not reentrant. That is, operations in one environment that are called from the other environment can't exercise the original environment, except to return data. In particular:

- The string passed to the BASIC environment by BASICX can't contain the keyword FORTHX. However, FORTH$, FORTHI, FORTHF are allowed.
- The FORTH command string that is the first argument of FORTHX can't contain the FORTH word BASICX. However, BASIC$, BASICI, and BASICF are allowed.

Applications that respect these two rules will work as long as operations in one environment respect the integrity of the other. For example, don't POKE random data into the FORTHRAM file from BASIC or write over the BASIC environment pointers from FORTH.

HP-IL Operations

To enable controller applications to take advantage of FORTH's speed, the FORTH kernel includes FORTH equivalents of the BASIC statements ENTER and OUTPUT. Additional HP-IL functionality in the FORTH environment can be gained by using the FORTH-to-BASIC words. For example, " STATUS" BASIC1 returns to the integer data stack a value describing the loop status.

The FORTH word ENTER instructs the HP 82401A HP-IL Interface to receive data from an HP-IL device. The HP-IL module puts the bytes received into a temporary location (the HP-71 math stack). The FORTH system then moves the bytes into an address specified by the user when executing ENTER. The byte count and the address of the data are always returned to the user.

If BASIC system flag −23 is set, ENTER terminates when it receives an End of Transmission message. Otherwise, ENTER continues to request data until its end condition is satisfied. The end condition can be either the reception of a specified number of bytes or of a particular byte value.

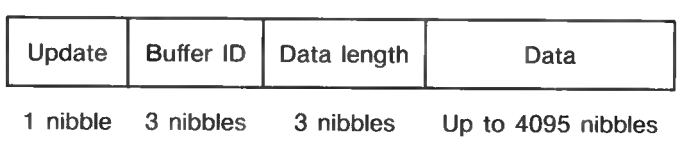
The FORTH word OUTPUT instructs the HP 82401A HP-IL Interface to send data to an HP-IL device. The user supplies a byte count and the address of the data to be output.

Two FORTH user variables, PRIMARY and SECONDARY, specify the intended device for OUTPUT and ENTER. Default contents of the variables are 1 for PRIMARY and 0 for SECONDARY. The user must ensure that these variables are properly set up before executing ENTER or OUTPUT.

General Purpose Buffers

Large applications may require blocks of temporary storage that are not a part of the FORTH dictionary space. The HP-71 BASIC O/S provides such temporary storage in the form of general purpose buffers. A maximum of 512 buffers can each contain a maximum of 4095 nibbles, provided that there is enough RAM present to allocate to the buffer. The FORTH dictionary provides five words to make, find, expand, contract and destroy these buffers.

General purpose buffers are maintained at the end of the file chain. The last general purpose buffer is followed by two zero bytes, signifying the end of the general purpose buffer chain. A general purpose buffer has a seven-nibble header field followed by the data space.



The update nibble is used by the operating system. Refer to the *HP-71 Software IDS* for a description.

Temporary buffers are allocated buffer ID's in the range of E00 to FFF. Because memory contents can move, shifting the position of the buffer, you must use the buffer ID to find the current location of the buffer each time you use it.

General purpose buffers are purged by the operating system at coldstart, power on, and during execution of `FREE PORT` and `CLAIM PORT`.

The following FORTH words deal with general purpose buffers.

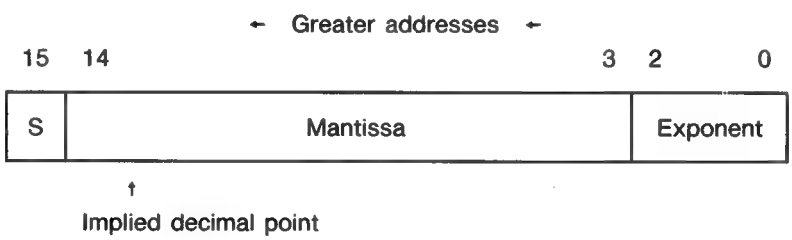
- `MAKEBF` creates a general purpose buffer of a specified size.
- `FINDBF` finds the current address of a specified general purpose buffer.
- `KILLBF` deletes a specified general purpose buffer.
- `EXPBF` expands a specified general purpose buffer by a specified number of nibbles.
- `CONBF` contracts a specified general purpose buffer by a specified number of nibbles.

FORTH Extensions

Floating-Point Operations

The HP-71 FORTH system includes an HP-RPN-style floating-point stack (X-, Y-, Z-, T-, and LAST X registers). There are FORTH words to manipulate the stack and to use the HP-71 math routines for floating-point operations. There are also FORTH words to create floating-point variables and constants, to fetch and store floating-point numbers, and to display floating-point numbers.

FORTH stores floating-point numbers in the same format as the BASIC system. Each register contains 16 nibbles, as shown below.

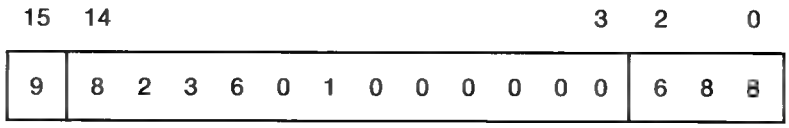


Sign. The sign nibble (labeled "S" above) contains 0 for a positive number and 9 for a negative number.

Mantissa. The 12-digit mantissa has an implied decimal point after the most significant digit. The mantissa is not necessarily normalized—that is, it can contain leading zeros to effectively extend the range of the exponent. This field may contain non-numeric data when the register contains an Inf or NaN.

Exponent. The three-digit exponent E is expressed in tens complement, $-499 \leq E \leq 499$, with the most significant digit in nibble 2. The exponent field is also used to indicate an Inf or NaN: F00 indicates Inf (which may be positive or negative), F01 indicates a quiet NaN, and F02 indicates a signaling NaN.

The following diagram shows how the number −8.23601 E−312 is stored in a register.



For more information about the formats for floating-point numbers, refer to the *HP-71 IDS*.

A floating-point number is identified in HP-71 FORTH input by the presence of a decimal point. When INTERPRET doesn't identify a character sequence in the input stream as a FORTH word, NUMBER checks the sequence for a decimal point. If there is no decimal point, NUMBER treats it as a potential single- or double-length number. (Many FORTH systems identify double-length numbers by the presence of ., .., :, /, or a non-leading −. HP-71 FORTH uses only .., :, and / to identify double-length numbers.)

If the sequence contains a decimal point, the entire sequence is passed to the BASIC O/S routine corresponding to the keyword VAL for evaluation. If the sequence can be evaluated, the result is pushed onto the floating-point stack. "Can be evaluated" means that the character sequence is any valid BASIC numeric expression, which may include literal numbers and BASIC numeric variables. For example, the sequence 10*SIN(30.) entered in the FORTH environment will return the value 5 to the floating-point X-register (assuming that the current HP-71 angular mode is degrees). Similarly, 1.*T1 will return the current value of the BASIC variable T1 to the X-register.

A side effect of the automatic floating-point expression evaluation is that attempted execution or compilation of unrecognized words containing decimal points will result in the BASIC message ERR:Data Type. For example, entering an undefined word XYZABC causes the FORTH message FTH ERR:XYZABC not recognized, but entering the XYZ.ABC will cause the BASIC message ERR:Data Type because of the decimal point.

Floating-point trigonometric functions use the current HP-71 angular mode. FORTH words are provided to switch the mode between degrees and radians. If the mode is set in FORTH, then subsequent BASIC operations will use that mode, and vice versa. Similarly, the floating-point display mode is common to FORTH and BASIC. Floating-point numbers are converted for output (F., FSTR$) in decimal according to the current display mode, which can be set from FORTH or BASIC.

The names of several floating-point operations are prefaced with "F" to distinguish them from operations with similar names. In the following description, x is the contents of the X-register, y is the contents of the Y-register, and so on. All floating-point arithmetic operations return the result to the X-register.

Floating-point Words

- F+ returns $y + x$.
- F− returns $y - x$.
- F* returns $y \times x$.
- F/ returns $y \div x$.
- HR Convers a from hours, minutes, seconds format (HH.MM.SSSS) to decimal hour.
- HMS converts x from decimal hours to hours, minutes, seconds format.

- HMS+ adds x+y in hours, minutes, seconds format.
- HMS- subtracts y−x in hours, minutes, seconds format.
- CLOCK returns the current HP-71 clock time in seconds.
- X^2 returns x^2.
- 10^X returns 10^x.
- SIN returns the sine of x.
- COS returns the cosine of x.
- TAN returns the tangent of x.
- E^X returns e^x.
- E^X-1 returns $e^{(x-1)}$.
- 1/X returns the reciprocal of x.
- SQRT returns the square root of x.
- Y^X returns y^x.
- LGT returns $\log_{10}$ of x.
- LN returns the natural log of x.
- LN1+X returns the natural log of $(x+1)$.
- ATAN returns the arc tangent of x.
- ASIN returns the arc sine of x.
- ACOS returns the arc cosine of x.
- RDN rolls down the stack ("down" in the HP-RPN sense).
- RUP rolls up the stack ("up" in the HP-RPN sense).
- X<>Y swaps x and y.
- X, Y, Z, T, and L return the address of the corresponding floating-point register.
- LASTX pushes the contents of the LAST X register onto the floating-point stack.
- FENTER pushes the contents of the X-register onto the floating-point stack.
- RCL fetches a floating-point number from the address on top of the data stack and pushes it onto the floating-point stack.
- STO stores x into the address on top of the data stack.
- F. displays x without altering the floating-point stack.
- FVARIABLE creates a floating-point variable in the FORTH dictionary.
- FCONSTANT creates a floating-point constant in the FORTH dictionary.
- X=0?, X<0?, X<=0?, X#0?, X>0?, X>=0?, X>Y?, X<Y?, X=Y?, X#Y?, X<=Y?, and X>=Y? perform the specified test and, if true, push a true flag (−1) onto data stack; or if false, push a false flag (0) onto data stack.
- DEGREES sets the active angular mode to degrees.
- GRAD sets the active angular mode to grads.
- RADIANS sets the active angular mode to radians.
- STD, FIX, ENG, and SCI set the display format.

- D-R converts x from degrees to radians.
- R-D converts x from radians to degrees.
- R-P converts the vector (x,y) from rectangular notation to polar (r,0).
- P-R converts a vector in polar notation (x = radius, y = angle) to rectangular notation (x,y).
- % returns $xy/100$.
- %CH returns $100((x-y)/y)$
- FACT returns $x!$.
- DEC converts x from decimal to octal.
- OCT converts x from octal to decimal.

String Operations

HP-71 FORTH includes words to create string constants, string variables, and string arrays; to compare strings; to manipulate portions of strings (substrings); and to match string patterns. A string is stored in memory in the following format.

Format of a String in Memory

Maximum length	Current length	Character string (left-justified)
1 byte	1 byte	Maximum-length bytes

A string in memory is usually represented on the stack by a pair of values: an address and a character count (count on top). The address is the location of the first character of the string in memory, and the character count is the current length. This is the format expected by the standard word TYPE.

Occasionally a "counted string" in memory is represented on the stack simply by an address. The address is the location of the string's length byte, which is followed in memory by the string's characters. This is the format expected by the standard word NUMBER.

String constants are created by the word ", which puts the maximum-length byte, the current-length byte, and the string in the pad (system scratch space). String constants are thus very temporary—don't type in two string constants followed by a comparison operator, because the second will have been created on top of the first. String constants are used mainly to set the values of string variables, but you can also use them with other functions as long as you notice when the pad is being overwritten.

String variables are dictionary entries much like numeric variables. At the PFA are the maximum-length and current-length bytes followed by the string. The code field contains the address of code that returns to the stack the address of the first character (PFA + 4) and the current length.

String variable arrays are similar to single variables, but the first two bytes at the PFA indicate the maximum length of each element and the number of elements in the array. Next come the strings, each in the format described above: maximum length, current length, string. The *n*th element is accessed by typing *n array name*; the CFA points to code that returns the address and count of this element, which can be manipulated just like a regular string variable or constant.

String Words

- " creates a temporary string.
- ASC returns the ASCII code for the first character in a string.
- CHR$ returns a temporary string of length 1 for a specified ASCII code.
- END$ creates a temporary substring from the last part of a string.
- FSTR$ converts the number in the X-register to a string.
- LEFT$ creates a temporary substring from the first part of a string.
- MAXLEN returns the maximum allocated length of a string.
- NULL$ creates a temporary string of zero length.
- POS returns the position within a string of a substring.
- RIGHT$ creates a temporary substring of specified length from the last part of a string.
- S= returns a true flag if two strings are equal, a false flag if not.
- S< returns a true flag if $string_1 < string_2$, a false flag if not.
- S! stores $string_1$ into $string_2$.
- S<& adds a copy of one string to the end of another string.
- S>& adds a copy of one string to the beginning of another string.
- SMOVE stores a string at a specified address.
- STR$ converts a double number into a string.
- STRING creates a string variable.
- STRING-ARRAY creates a string-array variable.
- SUB$ creates a temporary substring from the middle part of a string.

Vocabularies

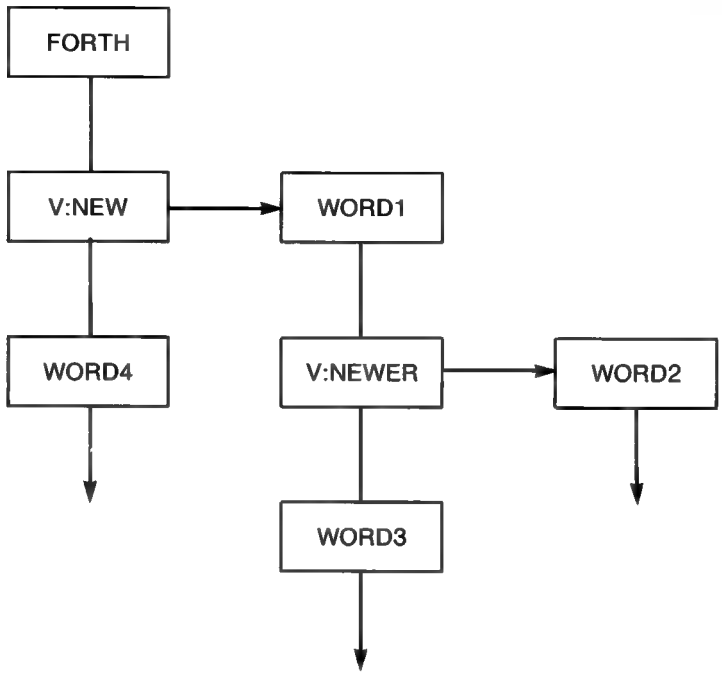
The HP-71 FORTH vocabulary structure is a tree-like structure. Every vocabulary contains the word FORTH, which sets the FORTH vocabulary as the CURRENT vocabulary (to which subsequent new words will be added). This is because FORTH is the first word in the FORTH vocabulary, and all vocabularies eventually chain back to the FORTH vocabulary. The following example creates a vocabulary called NEW.

```
VOCABULARY NEW
NEW DEFINITIONS
```

In the first line, VOCABULARY creates a new vocabulary called NEW. This entry, NEW, is entered into the current vocabulary, which is the FORTH vocabulary. Execution of NEW in the second line makes NEW the CONTEXT vocabulary (in which searches for words begin). DEFINITIONS sets the CURRENT vocabulary to be the same as the CONTEXT vocabulary. To continue the example:

```
: WORD1 ;
VOCABULARY NEWER
NEWER DEFINITIONS
: WORD2 ;
```

Now three vocabularies exist: FORTH, NEW, and NEWER. Suppose that WORD3 is added to the NEW vocabulary, and WORD4 is added to the FORTH vocabulary. The diagram below shows the result.



If either NEW or NEWER is the CONTEXT vocabulary, the word search won't find WORD4 in the FORTH vocabulary. If NEWER is the CONTEXT vocabulary, the word search won't find WORD3 in NEW, but it will find WORD1. In terms of the diagram, the word search proceeds in vocabularies other than the CONTEXT vocabulary by moving leftward and upward, never rightward or downward.

It is important to realize that, while FORTH can be reached from any vocabulary, the converse is not always true. NEW can be found when FORTH, NEW, or NEWER is the CONTEXT vocabulary, but NEWER can be found only when NEW or NEWER is the CONTEXT vocabulary.

Whenever an error occurs, FORTH becomes both the CONTEXT and CURRENT vocabulary.

The HP-41 Environment

The HP-41 Environment is an extension of the FORTH system. When the FTH41RAM file is created, the user dictionary initially contains two vocabulary words: FORTH and HP-41V (the latter is part of the former's word set). The HP-41V vocabulary contains all HP-41 postfix functions, and HP-41 functions that use the same name as standard FORTH words but operate differently (for example, ABS, SIGN, MOD).

HP-41 is a word in the FORTH vocabulary. When it is executed, the following occurs:

- HP-41V is set as the context and current vocabulary.
- A vectored form of INTERPRET is used, which carries out certain HP-41 emulator initializations, and executes a vectored display word (default—display the X-register).

- A vectored form of NUMBER is used, which sends all unrecognized input to the BASIC interpreter for potential evaluation as floating-point expressions.
- The dictionary is checked for the existence of the word R41, is used by the system as a marker. If the word does not exist, the user is prompted for Max SIZE?, and R41 is created in the dictionary. Space is alloted in the FTH41RAM file for the user-specified number of 8-byte floating-point registers.

Each HP-41 program compiled from a text file using LOAD is entered into the dictionary with a word name @*file name*, where *file name* is the name of the text file. Global and local labels are stored in the HP-41 label buffer. For this reason, you should not remove HP-41 programs from the dictionary using FORGET, which will delete the program but leave invalid entries in the label buffer.

When you execute FORTH from the HP-41 environment, the FORTH vocabulary is reset as the current and context vocabulary, and the default forms of INTERPRET and NUMBER are restored.

Relation to the HP-71 FORTH/Assembler ROM

The FORTH system contained in the HP-41 Translator Pac is very similar to that in the HP 82441A FORTH/Assembler ROM. The HP-41 Translator Pac does not contain an assembler, but its built-in word set contains additional floating-point words and other HP-41 words not found in the FORTH/Assembler ROM.

The floating point words in the HP-41 Translator Pac differ from their FORTH/Assembler ROM counterparts in two respects:

- The HP-41 Translator Pac words will error if any of their arguments are alpha data.
- The HP-41 Translator Pac words perform a stack lift only after computation is finished, so that the stack is unaffected if a word terminates due to an error. In the FORTH/Assembler ROM, the stack lift is performed before the computation.

FORTH words defined from words common to both modules can be compiled into either system from a text file containing the definitions. However, because the addresses of built-in words and user variables are not the same in the two systems, you should not attempt to exchange files of type FORTH between the systems. That is, a file originally created as FORTHRAM by the FORTH/Assembler module should not be renamed as FTH41RAM for use with the HP-41 Translator Pac, and vice-versa. The most likely result of such an exchange is the Memory Lost error.

Error Trapping

When an error occurs during execution of a FORTH word, a system routine equivalent to ABORT or ABORT" is executed. Normally, these routines will reset the data and return stacks and return to the outer interpreter loop for new input. However, HP-71 FORTH provides an error-trapping facility that can allow FORTH execution to continue after an error.

The user variable ONERR contains the CFA of a word to execute when an error occurs. The system abort routines check the contents of ONERR; if ONERR contains zero, the routines will exit normally through QUIT. If the value of ONERR is non-zero, execution will be transferred to the address contained in ONERR. The stacks are not reset, so the error routine has a chance to recover some or all of the state of the system at the time of the error.

FORTH Memory Organization

HP-71 Memory

The diagram below shows a map of the HP-71 memory with the HP-41 Translator Pac installed.

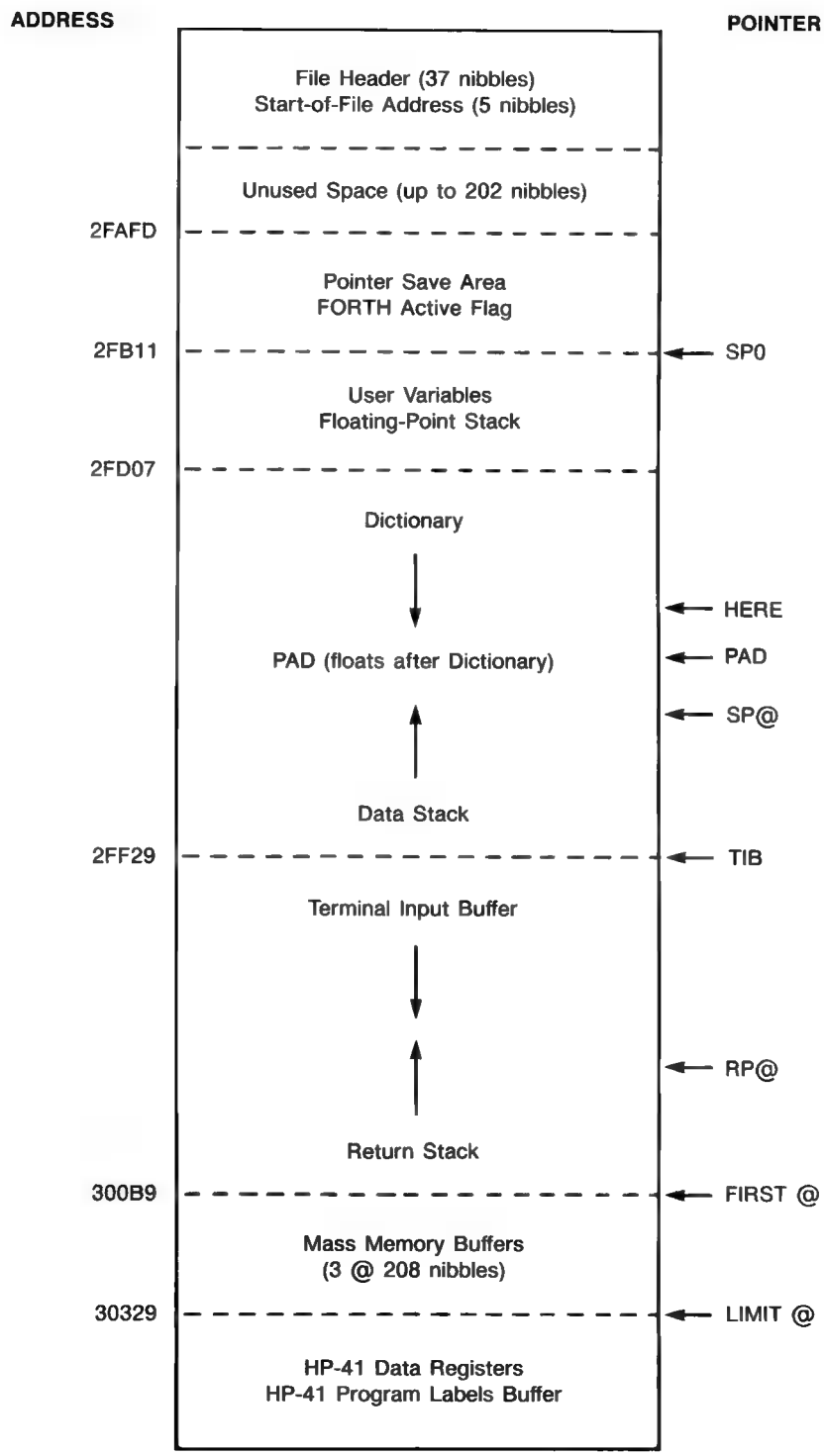


Figure 5-1. FTH41RAM File Structure

The HP-41 Translator Pac ROM uses addresses in three regions:

- Hard-configured ROM, from E0000 to EFFFF. The hard-configured ROM contains the FORTH operating system, the built-in FORTH dictionary, and the HP-41 emulator.
- Soft-configured ROM. This is a 16K-byte module that contains the editor, all BASIC keywords in the HP-41 Translator ROM, and the initialization routines for the FORTH and HP-41 environments.
- The FTH41RAM file. This file is stored in user memory and contains the changeable parts of the HP-41/FORTH environments—user variables, user dictionary, and so on. When the FORTH or HP-41 environment is active, FTH41RAM will always be the first file in user memory.

The FTH41RAM File

When FORTH, FORTHX, or HP41 is executed from the BASIC environment, a file called FTH41RAM is created (unless it exists already). FTH41RAM contains both the FORTH system's status information and all words added by users. FORTH has been assigned LIF file types E218 and E219. When the HP-41 Translator ROM is plugged in and a CATALL is executed, the FORTH system intercepts the file-type poll and displays FORTH instead of the numeric file type for FTH41RAM. Initially FTH41RAM contains about 1K byte. You can enlarge the file to provide user dictionary space after the entire 1K-byte file exists.

There are four words in the dictionary that allow you to control the amount of dictionary space (in nibbles) available for new definitions:

- GROW increases dictionary space by the number specified on the data stack.
- SHRINK decreases dictionary space by the number specified on the data stack.
- DSIZE sets the dictionary space equal to the number specified on the data stack.
- XSIZE sets the dictionary space equal to the (integer part of the) number specified in the X-register.

To re-enter FORTH when FTH41RAM is no longer the first file in memory, 37 bytes are required to swap the file back into the first position. If there is not enough memory, an error message is displayed.

Copying FTH41RAM. You can rename, copy, and purge FTH41RAM using HP-71 BASIC file commands. This enables you to have multiple versions of the FORTH system, each containing a different user dictionary. When you have multiple FORTH files, the file currently named FTH41RAM will be the active FORTH file when you enter the FORTH environment. Also, if you make backup copies of your FORTH system, you can restore your system following a memory loss (common when programming in FORTH) by reloading a FTH41RAM file from mass storage rather than by recompiling the dictionary. The HP-41 Pac is not required to copy the FTH41RAM file out to mass storage, but it is required to copy FTH41RAM back into RAM.

Contents of FTH41RAM. Figure 5-1 shows the structure of FTH41RAM. At the beginning of the file are 37 nibbles of system overhead—file name, file type, link to next file, and so on. Next is the address of the FTH41RAM file; when the FORTH system is re-entered, this address indicates whether FTH41RAM has been moved. Next is up to 101 bytes of unused space, depending on FTH41RAM's starting address. Enough space is added to ensure that FTH41RAM's data begins at 2FAFD.

Starting at 2FAFD is the housekeeping information needed to save the FORTH pointers when a system routine alters all of the CPU registers. At 2FB11 starts the block of FORTH system variables called "user variables." The floating-point stack follows the user variables in the file. The user dictionary space starts above the floating-point stack. The data stack is deep enough to hold a minimum of 40 entries. The return stack and the Terminal Input Buffer share 200 bytes, of which a maximum of 98 bytes can be used by the Terminal Input Buffer (keyboard entry is limited to 96 characters, and FORTH appends 2 null characters for its own use). The mass memory buffers are allocated 312 bytes.

The last entry in the FTH41RAM file is the HP-41 label buffer, which contains the global and local labels from all HP-41 programs contained in the FORTH dictionary. The buffer has a minimum size of 5 bytes.

The tables below show the details of a newly created FTH41RAM file. Although the FTH41RAM file is always the first file in user memory, its starting address varies according to the length of the HP-71 configuration buffers, which precede FTH41RAM in memory. The current address of the start of the file can be found by executing

```
ADDR$('FTH41RAM') in BASIC, or
" FTH41RAM" FINDF in FORTH.
```

Table 5-2. System Save Area

Address	Contents
2FAFD	Data-stack pointer save.
2FB02	Return-stack pointer save.
2FB07	Instruction pointer save.
2FB0C	FORTH active flag.

Table 5-3. User Variables

Address	Contents	FORTH Words To Return Contents
2FB11	Pointer to bottom of data stack.	S0 or SP0 @
2FB16	Pointer to bottom of return stack.	RP0 @
2FB1B	Pointer to TIB.	TIB
2FB20	Next buffer.	USE @
2FB25	Most recent mass storage buffer.	PREV @
2FB2A	First mass storage buffer.	FIRST @
2FB2F	End of FTH41RAM + 1.	LIMIT @
2FB34	Vocabulary link.	
2FB39	Buffer record size.	
2FB3E	Number of characters in TIB.	#TIB @
2FB43	Maximum word-name length.	WIDTH @
2FB48	Warning mode.	WARN @
2FB4D	Enable/disable OK in QUIT.	OKFLG @

Table 5-3. User Variables (Continued)

Address	Contents	FORTH Words To Return Contents
2FB52	Line number in current LOADF file. (Reset when load error occurs.)	BLK @
2FB57	Offset in TIB.	>IN @
2FB5C	Number of characters read by EXPECT96.	SPAN @
2FB61	FIB# of active LOADF file.	SCRFIB @
2FB66	Address of CONTEXT vocabulary.	CONTEXT @
2FB6B	Address of CURRENT vocabulary.	CURRENT @
2FB70	Compilation flag.	STATE @
2FB75	Current base.	BASE @
2FB7A	Number type indicator.	
2FB7F	Unused. Available for user programming.	
2FB84	Current position of stack. (Used by compiler.)	
2FB89	Pointer to last character in display string.	
2FB8E	FORGET boundary.	FENCE @
2FB93	Next available nibble in dictionary.	
2FB98	Buffer size in nibbles.	
2FB9D	Line number in current LOADF file. (Preserved after load error.)	LINE# @
2FBA2	Return address for BASIC keywords.	
2FBA7	Reserved for HP-IL use.	
2FBAC	Secondary HP-IL address.	SECONDARY @
2FBB1	Primary HP-IL address.	PRIMARY @
2FBB6	On-error execution address.	ONERR @
2FBBB	Error-occurence flag.	

Table 5-4. Floating-Point Stack Registers

Address	Contents	FORTH Words To Return Value to X-register
2FBC0	LAST X register.	L RCL
2FBD0	X-register.	X RCL
2FBE0	Y-register.	Y RCL
2FBF0	Z-register.	Z RCL
2FC00	T-register.	T RCL
2FC10	System use. (Eight bytes for file name.)	

Table 5-5. Vectored Execution Addresses

Address	Contents
2FC20	INTERPRET
2FC25	CREATE
2FC2A	NUMBER
2FC2F	, (*comma*)
2FC34	C, (*c-comma*)
2FC39	ALLOT
2FC3E	For *xxx* isn't unique message.

Table 5-6. HP-41 Emulator User Variables

Address	Contents
2FC43	Emulator active flag
2FC48	HP-41 program pointer and return stack
2FC70	HP-41 flags
2FC80	Alpha register
2FCE4	Maximum register number or SIZE −1
2FCE9	Sigma register number
2FCEE	Vector to HP-41 display word
2FCF3	HP-41 program return stack level pointer
2FCF8	Scratch area
2FC02	FORTH program status

Table 5-7. User Dictionary and Above

Address	Contents	FORTH Words To Return Contents
2FD07	FORTH word.	
2FD2B	HP-41V vocabulary word.	
2FD54	HP-41V null word.	
2FD5F	Start of first user-defined word. (Addresses above 2FCB1 are variable.)	
2FD5F*	End of dictionary. (Next available nibble.)	HERE
2FDB9*	Pad. (Floats after dictionary.)	PAD
2FF29†	Top of data stack.	SP@
2FF29†	Bottom of data stack = Start of TIB.	S0, SP0 @, or TIB
300B9†	Bottom of return stack = Start of first mass storage buffer.	RP0 @ FIRST @
30329†	HP-41 data registers and labels buffer.	LIMIT @

* Changes when words are compiled.
† Changes when GROW or SHRINK is executed.

The FORTH Dictionary

When you type in a word to be executed or when the system compiles a word from a source file, FORTH must search through its dictionary to find the word and its execution address. HP-71 FORTH searches the RAM part of the dictionary first (the user dictionary) and then the ROM part (the built-in FORTH words). Words in ROM are arranged according to word length to minimize the search time. The length of the target word is used as an index into a jump table so that, for example, only the list of three-character words are searched for a three-character word. A test is also made to ensure that the word is not longer than the longest word in the ROM portion of the dictionary.

As an example of an entry in the dictionary, the structure of a FORTH primitive CMOVE is shown below. Although this word is in the ROM dictionary, its structure is typical of words in either the ROM or RAM parts of the dictionary.

Table 5-8. Structure of a Word

Field	Address	Contents
Link	LFA = E43F2	E43AA
Name	NFA = E43F7	5834D4F4655C
Code	CFA = E4403	E4408
Parameter	PFA = E3B04	*code*

Link Field (LFA). The contents of the link field (E43AA) point to the name field of the previous dictionary entry.

Name Field (NFA). The first byte of the name field, 85, is 10000101 in binary (note that the byte's two nibbles are reversed, with "5" stored at a smaller address than "8"). The byte's high-order bit is set to indicate the start of the name field, and the second bit is clear to indicate that the word is not immediate. The third bit (the smudge bit, set during compilation of a secondary to prevent the word being used in its own definition) is clear. The five low-order bits have a value of 5 to indicate that the name is five characters long; the maximum length is 31 characters. The second and subsequent bytes in the name field are the ASCII representation of the word's name, with the high bit of the last character is set to indicate the end of the name field. Here the last character is "E" with ASCII value 01000101, so the binary value 11000101 is stored (with nibbles reversed) as 5C.

Code Field (CFA). Because CMOVE is a primitive, the code field contains this word's PFA, E4408, so that the code in the parameter field will be executed. In a secondary, the code field contains the address of the run-time code of :, which nests the FORTH program pointer down one level.

Parameter Field (PFA). Because CMOVE is a primitive, the parameter field contains executable code. In a secondary, the parameter field contains the CFAs of the words that make up the secondary.

The ROM-based dictionary contains all of the built-in FORTH words except FORTH, which is always the first word in the RAM-based dictionary. To speed compilation, the FORTH system doesn't search the entire ROM-based dictionary. The ROM-based dictionary is composed of 13 separate linked lists, with each list containing words of a specific length, so the FORTH system searches only the list for the appropriate word length.

At E0000 is a jump table with 13 entries. Each entry contains a pointer to the beginning of the word list for words of a specific length, from 0 through 12 characters. To illustrate this structure, a word VLIST appears below that will display all words in the ROM dictionary. Note that the pointer initially indicates the list of one-character words.

```
HEX
: VLIST E0005
 0 1 DO
  DUP @
   BEGIN DUP
    COUNT 1F AND 1-
    DUP >R 2* SWAP DUP >R
    + C@ 7F AND R> R>
    TYPE EMIT CR 5- @ ?DUP 0=
   UNTIL
  5+ LOOP DROP ;
```

The HP-71 File System

The HP-71 contains a 64K-byte operating system kernel that starts at address 00000. The kernel performs various control functions and contains the BASIC interpreter. External software may be added to the machine in the form of files that the kernal interprets or executes directly. These files may be directly plugged into the machine through ROM or RAM modules, or copied into the machine from external media such as cards or tape.

File Types

The following file types are directly supported by the HP-71 mainframe. OEM software developers may support other file types by first reserving the file type with Hewlett-Packard and then including the appropriate poll handlers in a LEX file. Each file type is identified by a 16-bit value that conforms to Hewlett-Packard's Logical Interchange Format for Mass Media.

When HP-71 files are stored on external media, file security and privacy are encoded, if applicable, in the numeric file type as shown in the chart below. When files are stored in memory, privacy and security are encoded in the flags field of the file header, and the file type stored in the file header is *always* the normal file type.

Table 5-9. Numeric File Type

Type	Description	Normal	Secure	Private	Execute Only
BASIC	Tokenized BASIC program.	E214	E215	E216	E217
BIN	HP-71 machine language.	E204	E205	E206	E207
DATA	Fixed data.	E0F0	E0F1	n/a	n/a
LEX	Language extension.	E208	E209	E20A	E20B
KEY	Key assignment.	E20C	E20D	n/a	n/a
SDATA	Stream data.	E0D0	n/a	n/a	n/a
TEXT	ASCII text, in LIF Type 1 format.	0001	E0D5	n/a	n/a
FORTH	FORTH file.	E218	E219	n/a	n/a

Four of these file types are program files: BASIC, BIN (Binary), LEX (Language Extension), and FORTH. BASIC files can be developed on the HP-71 using the built-in BASIC interpreter. FORTH files can be developed using the HP-41 Translator ROM. BIN, LEX, and FORTH files can be developed on the HP-71 using the FORTH/Assembler ROM.

Table 5-10. Types of Program Files

Type	Format	Method of Invocation	Mode of Execution
BASIC	Tokenized BASIC statements.	RUN or CALL command.	Interpretation.
BIN	Machine language (binary).	RUN or CALL command.	Direct execution.
LEX	Language extension file; adds BASIC keywords, messages, and functional extensions; written in machine language.	Through its added BASIC keywords and by polls from operating system.	Direct execution.
FORTH	FORTH vocabulary.	Through FORTH interpreter.	Threaded interpretation.

Structure of the File Chain

The HP-71 maintains a file area in main RAM that is composed of a linked list, or chain, of file entries. (Each plug-in ROM module and independent RAM contains its own file chain.) At the beginning of each file entry is a file header. The file header contains identifying information about the file along with the link to the next file entry in the chain. The end of the chain is marked by a zero byte. Each file header contains the following fields:

Table 5-11. Fields in a File Header

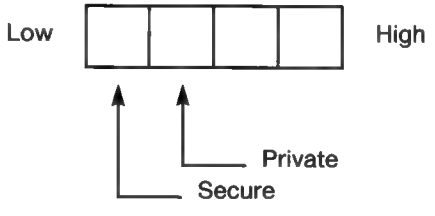
Field	Size
File name	16 nibbles
File type	4 nibbles
Flag	1 nibble
Copy Code	1 nibble
Creation Time	4 nibbles
Creation Date	6 nibbles
Link	5 nibbles

File Name. The file-name field contains the eight-character file name in ASCII, filled with blanks to the right (high memory).

File Type. The file-type field contains a four-digit hex integer, listed in the "File Types" table above.

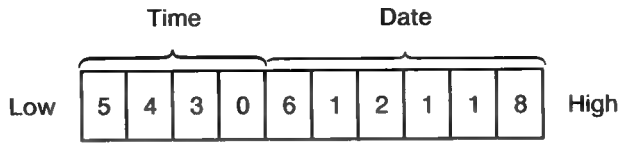
Flag. The flag field contains four system flags. The two bits in the low end of the flag field indicate file protection. When set, the lower of the two bits indicates a file is SECURE; the higher of the two bits indicates a file is PRIVATE. The remaining two bits of the flag field are unused.

File Header-Flags



Copy Code. The copy-code field indicates the file attributes neccessary for external copying.

Creation Time and Creation Date. The creation-time and creation-date fields represent the time and date in BCD. The time field contains four nibbles; the minutes are in the low byte, and the hour is in the high byte. The date field contains six nibbles; the day is represented in the low byte, the month in the next byte, and the year in the high byte. For example, the internal representation of 03:45 on December 16, 1981, would be as follows:



Link. The link field contains the offset to the next file (header) in memory.

Appendix A

Care, Warranty, and Service Information

Care of the Module

The HP 82490A HP-41 Translator Pac module does not require maintenance. However, there are several precautions, listed below, that you should observe.

CAUTIONS

- Do not place fingers, tools, or other objects into the plug-in ports. Damage to plug-in module contacts and the computer's internal circuitry may result.
- Turn off the computer (press [f][OFF]) before installing or removing a plug-in module.
- If a module jams when inserted into a port, it may be upside down. Attempting to force it further may result in damage to the computer or the module.
- Handle the plug-in modules very carefully while they are out of the computer. Do not insert any objects in the module connector socket. Always keep a blank module in the computer port when a module is not installed. Failure to observe these cautions may result in damage to the module or the computer.

Limited One-Year Warranty

What We Will Do

The HP-41 Translator Pac module is warranted by Hewlett-Packard against defects in materials and workmanship affecting electronic and mechanical performance, but not software content, for one year from the date of original purchase. If you sell your unit or give it as a gift, the warranty is transferred to the new owner and remains in effect for the original one-year period. During the warranty period, we will repair or, at our option, replace at no charge a product that proves to be defective, provided you return the product, shipping prepaid, to a Hewlett-Packard service center.

What Is Not Covered

This warranty does not apply if the product has been damaged by accident or misuse or as the result of service or modification by other than an authorized Hewlett-Packard service center.

No other express warranty is given. The repair or replacement of a product is your exclusive remedy. **ANY OTHER IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS IS LIMITED TO THE ONE-YEAR DURATION OF THIS WRITTEN WARRANTY.** Some states, provinces, or countries do not allow limitations on how long an implied warranty lasts, so the above limitation may not apply to you. **IN NO EVENT SHALL HEWLETT-PACKARD COMPANY BE LIABLE FOR CONSEQUENTIAL DAMAGES.** Some states, provinces, or countries do not allow the exclusion or limitation of incidental or consequential damages, so the above limitation or exclusion may not apply to you.

This warranty gives you specific legal rights, and you may also have other rights which vary from state to state, province to province, or country to country.

Warranty for Consumer Transactions in the United Kingdom

This warranty shall not apply to consumer transactions and shall not affect the statutory rights of a consumer. In relation to such transactions, the rights and obligations of Seller and Buyer shall be determined by statute.

Obligation to Make Changes

Products are sold on the basis of specifications applicable at the time of manufacture. Hewlett-Packard shall have no obligation to modify or update products once sold.

Warranty Information

If you have any questions concerning this warranty, please contact an authorized Hewlett-Packard dealer or a Hewlett-Packard sales and service office. Should you be unable to contact them, please contact:

- In the United States:

Hewlett-Packard
Personal Computer Group
Customer Support
11000 Wolfe Road
Cupertino, CA 95014

Toll-Free Number: (800) FOR-HPPC (800 367-4772)

- In Europe:

Hewlett-Packard S.A.
150, route du Nant-d'Avril
P.O. Box CH-1217 Meyrin 2
Geneva
Switzerland
Telephone: (022) 83 81 11

Note: Do *not* send units to this address for repair.

- In other countries:

Hewlett-Packard Intercontinental
3495 Deer Creek Rd.
Palo Alto, California 94304
U.S.A.
Telephone: (415) 857-1501

Note: Do not send units to this address for repair.

Service

Hewlett-Packard maintains service centers in most major countries throughout the world. You may have your unit repaired at a Hewlett-Packard service center any time it needs service, whether the unit is under warranty or not. There is a charge for repairs after the one-year warranty period.

Hewlett-Packard products are normally repaired and reshipped within five (5) working days of receipt at any service center. This is an average time and could vary depending upon the time of year and the work load at the service center. The total time you are without your unit will depend largely on the shipping time.

Obtaining Repair Service in the United States

The Hewlett-Packard United States Service Center for battery-powered computational products is located in Corvallis, Oregon:

Hewlett-Packard Company
Service Department

P.O. Box 999
Corvallis, Oregon 97339, U.S.A.
or
1030 N.E. Circle Blvd.
Corvallis, Oregon 97330, U.S.A.

Telephone: (503) 757-2000

Obtaining Repair Service in Europe

Service centers are maintained at the following locations. For countries not listed, contact the dealer where you purchased your unit.

AUSTRIA
HEWLETT-PACKARD Ges.m.b.H.
Kleinrechner-Service
Wagramerstrasse-Lieblgasse 1
A-1220 Wien (Vienna)
Telephone: (0222) 23 65 11

BELGIUM
HEWLETT-PACKARD BELGIUM SA/NV
Woluwedal 100
B-1200 Brussels
Telephone: (02) 762 32 00

DENMARK
HEWLETT-PACKARD A/S
Datavej 52
DK-3460 Birkerod (Copenhagen)
Telephone: (02) 81 66 40

EASTERN EUROPE
Refer to the address listed under Austria.

FINLAND
HEWLETT-PACKARD OY
Revontulentie 7
SF-02100 Espoo 10 (Helsinki)
Telephone: (90) 455 02 11

FRANCE
HEWLETT-PACKARD FRANCE
Division Informatique Personnelle
S.A.V. Calculateurs de Poche
F-91947 Les Ulis Cedex
Telephone: (6) 907 78 25

GERMANY
HEWLETT-PACKARD GmbH
Kleinrechner-Service
Vertriebszentrale
Berner Strasse 117
Postfach 560 140
D-6000 Frankfurt 56
Telephone: (611) 50041

ITALY
HEWLETT-PACKARD ITALIANA S.P.A.
Casella postale 3645 (Milano)
Via G. Di Vittorio, 9
I-20063 Cernusco Sul Naviglio (Milan)
Telephone: (2) 90 36 91

NETHERLANDS
HEWLETT-PACKARD NEDERLAND B.V.
Van Heuven Goedhartlaan 121
NL-1181 KK Amstelveen (Amsterdam)
P.O. Box 667
Telephone: (020) 472021

NORWAY
HEWLETT-PACKARD NORGE A/S
P.O. Box 34
Oesterndalen 18
N-1345 Oesteraas (Oslo)
Telephone: (2) 17 11 80

SPAIN
HEWLETT-PACKARD ESPANOLA S.A.
Calle Jerez 3
E-Madrid 16
Telephone: (1) 458 2600

SWEDEN
HEWLETT-PACKARD SVERIGE AB
Skalholtsgatan 9, Kista
Box 19
S-163 93 Spanga (Stockholm)
Telephone: (08) 750 2000

SWITZERLAND
HEWLETT-PACKARD (SCHWEIZ) AG
Kleinrechner-Service
Allmend 2
CH-8967 Widen
Telephone: (057) 31 21 11

UNITED KINGDOM
HEWLETT-PACKARD Ltd
King Street Lane
GB-Winnersh, Wokingham
Berkshire RG11 5AR
Telephone: (0734) 784 774

International Service Information

Not all Hewlett-Packard service centers offer service for all models of HP products. However, if you bought your product from an authorized Hewlett-Packard dealer, you can be sure that service is available in the country where you bought it.

If you happen to be outside of the country where you bought your unit, you can contact the local Hewlett-Packard service center to see if service is available for it. If service is unavailable, please ship the unit to the address listed above under Obtaining Repair Service in the United States. A list of service centers for other countries can be obtained by writing to that address.

All shipping, reimportation arrangements, and customs costs are your responsibility.

Service Repair Charge

There is a standard repair charge for out-of-warranty repairs. The repair charges include all labor and materials. In the United States, the full charge is subject to the customer's local sales tax. In European countries, the full charge is subject to Value Added Tax (VAT) and similar taxes wherever applicable. All such taxes will appear as separate items on invoiced amounts.

Computer products damaged by accident or misuse are not covered by the fixed repair charges. In these situations, repair charges will be individually determined based on time and materials.

Service Warranty

Any out-of-warranty repairs are warranted against defects in materials and workmanship for a period of 90 days from date of service.

Shipping Instructions

Should your unit require service, return it with the following items:

- A completed Service Card, including a description of the problem.
- A sales receipt or other proof of purchase date if the one-year warranty has not expired.

The product, the Service Card, a brief description of the problem, and (if required) the proof of purchase date should be packaged in adequate protective packaging to prevent in-transit damage. Such damage is not covered by the one-year limited warranty; Hewlett-Packard suggests that you insure the shipment to the service center. The packaged unit should be shipped to the nearest Hewlett-Packard designated collection point or service center. Contact your dealer for assistance. (If you are not in the country where you originally purchased the unit, refer to "International Service Information" above.)

Whether the unit is under warranty or not, it is your responsibility to pay shipping charges for delivery to the Hewlett-Packard service center.

After warranty repairs are completed, the service center returns the unit with postage prepaid. On out-of-warranty repairs in the United States and some other countries, the unit is returned C.O.D. (covering shipping costs and the service charge).

Further Information

Circuitry and designs are proprietary to Hewlett-Packard, and service manuals are not available to customers. Should other problems or questions arise regarding repairs, please call your nearest Hewlett-Packard service center.

When You Need Help

Hewlett-Packard is committed to providing after-sale support to its customers. To this end, our customer support department has established phone numbers that you can call if you have questions about this product.

Product Information. For information about Hewlett-Packard dealers, products, and prices, call the toll-free number below:

(800) FOR-HPPC
(800 367-4772)

Technical Assistance. For technical assistance with your product, call the number below:

(503) 757-2004

For either product information or technical assistance, you can also write to:

Hewlett-Packard
Portable Computer Division
Customer Technical Support
1000 N.E. Circle Blvd.
Corvallis, OR 97330

Appendix B

Error Messages

The error messages listed in the following tables relate only to HP-41 Translator Pac operations. For other error or warning messages, refer to the *HP-71 Reference Manual.*

This appendix contains three listings:

1. An alphabetical listing of HP-41/FORTH error messages with their corresponding error numbers. You can use the error's number to look up the error in the next listing.
2. A numerical listing of HP-41/FORTH error messages with a description of each error condition.
3. An alphabetical listing of editor messages with a description of each message.

FORTH Messages

Alphabetical Listing of HP-41/FORTH Messages

Message	Number
Address Not Inside a File	47084
Alpha Data	47096
Argument < 1	47070
Attempted to Redefine Null	47075
Bad Parameters	47086
BASIC Not Re-entrant	47094
Cannot Load	47090
Compile Only	47073
Conditionals Not Paired	47081
Configuration	47087
Data Error	47099
Definition Not Finished	47071
Dictionary Full	47072
Empty Stack	47078
FORTH Not Re-entrant	47095
FTH41RAM File Not in Place	47082
Full Stack	47079
HP-IL Error	47074
Illegal CASE Structure	47092
In Protected Dictionary	47077
Insufficient Memory	47097
Invalid Filespec	47083
No DO Before LEAVE	47091
No Ending "	47069
No Ending)	47068
No Ending ;	47067
Nonexistent	47098
No Printer	47100

Alphabetical Listing of HP-41/FORTH Messages (Continued)

Message	Number
____ Not Found	47066
Not in Current Vocabulary	47089
Not Programmable	47101
____ Not Recognized	47080
String Won't Fit	47088

Numerical Listing of HP-41/FORTH Messages with Descriptions

Error Number	Message and Condition
47066	____ Not Found The argument to ' (tick) isn't in the dictionary. Check the spelling of the word.
47067	No Ending ; The definition being compiled from a text file is unfinished. Put in an ending semicolon.
47068	No Ending) .(or (isn't matched by an ending parenthesis. Put in an ending parenthesis.
47069	No Ending " ." or " isn't matched by an ending double quote. Put in an ending double quote.
47070	Argument < 1 A word that expects positive integers finds negative numbers or zero on the stack. Ensure the proper values on the stack.
47071	Definition Not Finished The stack's size at the end of a word doesn't equal its size at the start. Review the control structures and immediate words used in the definition.
47072	Dictionary Full The dictionary space in FTH41RAM is used up. Use FORGET or GROW.
47073	Compile Only A compile-time word is used at run time. Check word usage in definitions.
47074	HP-IL Error Something is wrong related to the HP-IL interface. Check that the HP-IL interface is plugged into the HP-71; check the integrity of the loop.
47075	Attempted to Redefine Null A colon (starting a colon definition) is the only input received from the keyboard; or WORD '' or WORDI '' appears in a primitive assembly. Fatal to assembly. You can't redefine the null word in FORTH.
47077	In Protected Dictionary The argument for FORGET is below FENCE (or in ROM). Reset FENCE.
47078	Empty Stack A word expecting stack parameters finds the stack empty. Provide stack parameters.
47079	Full Stack The space in FTH41RAM for the data stack is used up. Use GROW to enlarge FTH41RAM or use FORGET to make space in FTH41RAM.

Error Number	Message and Condition
47080	____ Not Recognized The input is neither an existing word nor a number. Check the spelling of the word; check the CONTEXT vocabulary.
47081	Conditionals Not Paired A control-structure word (such as THEN) appears without the preceding word (such as IF). Supply the missing word.
47082	FTH41RAM File Not in Place FORTHI, FORTHF, or FORTH$ is attempted when the FTH41RAM file hasn't been created or has moved. Use FORTH or FORTHX to enter FORTH and then exit.
47083	Invalid Filespec The argument to FINDF is an illegal file specifier. Supply a valid file specifier.
47084	Address not inside a file ADJUSTF is given an address not properly within a file, such as the address of a file header. Check the address of the file.
47086	Bad Parameters A string word finds an out-of-range value on the stack, such as a character-position parameter of 20 for a string only 10 characters long. Check the stack value.
47087	Configuration An oversized configuration buffer or an erroneous pointer to that buffer prevents the FORTHRAM file from occupying its required location. This will never occur under normal circumstances. Remove a LEX file from RAM or remove a module.
47088	String Won't Fit A string is too long for the specified variable. Check the size of the variable.
47089	Not in Current Vocabulary The argument for FORGET isn't in the CURRENT vocabulary. Check the spelling of the word and the CURRENT vocabulary.
47090	Cannot Load The file is open, doesn't exist, etc. Check the file's status.
47091	No DO Before LEAVE LEAVE is used outside a DO-loop. Use LEAVE only inside a DO-loop.
47092	Illegal CASE Structure ENDCASE isn't preceded by valid CASE . . . OF . . . ENDOF structure. Check the complete control structure.
47094	BASIC Not Re-entrant BASICX is used in an argument to FORTHX. Eliminate such usage.
47095	FORTH Not Re-entrant FORTHX is used in an argument to BASICX, or in a program or user-defined function executed from BASICX, BASICI, or BASICF. Eliminate such usage.
47096	Alpha Data Alpha data was used for a function requiring numeric input.
47097	Insufficient Memory There is insufficient memory available to increase the FORTH dictionary by the amount specified with SIZE, DSIZE, PSIZE, or XSIZE, or to load a file into HP-41 emulator memory. Refer to Creating New Functions on page 47 for instructions on increasing the size of the FTH41RAM file.

Error Number	Message and Condition
47098	Nonexistent A register used by an HP-41 register function, or a label specified with GTO or XEQ, does not exist. Check the function syntax; compare numeric labels to SIZE?.
47099	Data Error An invalid parameter is supplied for an HP-41 function:

Function:	Condition:
TONE	$x > 9$
FIX, SCI, ENG	$x > 11$
AROT, POSA, XTOA, X<>F	$x > 255$
CLRGX, PRREGX	$x > 999$
STOFLAG	x not obtained with RCLFLAG $x > 43$ if y was obtained RCLFLAG
X=NN?, X≠NN?, X<NN?, X<=NN?, X>NN?, X>=NN	y contains alpha data other than "X", "Y", "Z", "T", or "L".
S+, S-	X, Y, or any of the summation registers contains Inf or NaN
OCT	$\|x\| > 68719476735$ or x is a non-integer or NaN.
DEC	$\|x\| > 777777777777$, x is a non-integer or NaN, or x contains a non-octal digit.

47100	No Printer An HP-41 printer function is executed when flag 55 is clear, indicating that no printer is present. Check printer connection and status of flag 55 (FS? 55).
47101	Not Programmable LOAD is executed with an intermediate file containing one of the functions: HP41, CLP, LOAD, RUN, or PRINTER.

Editor Messages

DONE

The editor has been exited.

File Exists: ____

The file specified to receive deleted lines already exists. Use the ± option, or choose a different filename.

Insufficient Memory

There is insufficient memory for the operation being performed. If other operations requiring less memory can be performed, the Cmd: prompt returns to the display. If no further operations are possible, the editor is exited. Purge a file or execute DESTROY ALL.

Invalid File Type: ____

The file specified in the command string must be a text file.

Invalid Param: ____

The editor doesn't recognize the parameter portion of a command string. Review the command's syntax.

Line Too Long

The line of text is longer than 96 characters, which is not allowed in text mode.

? Cmd: ____

The editor doesn't recognize the letter as a valid command. The valid commands are c, d, e, f, h, i, l, m, p, r, s, and t.

Working...

The editor is executing a command.

Appendix C

BASIC Keywords

Introduction

This appendix describes the BASIC keywords added to the HP-71 when the HP-41 Translator Pac module is plugged in. The keywords fall into three categories:

BASIC-to-FORTH	Editor	BASIC/FORTH-to-HP-41
FORTH	DELETE#	HP41
FORTH$	EDTEXT	
FORTHF	FILESZR	
FORTHI	INSERT#	
FORTHX	MSG$	
	REPLACE#	
	SCROLL	
	SEARCH	

Organization

Entries in this appendix are arranged in alphabetical order. The same format is used for every keyword entry so that you can quickly find the information you need. The format is similar to that used in the *HP-71 Reference Manual*—refer to that manual for additional details.

Each keyword entry provides the following information for the keyword:

- **Keyword name.** Shows the basic keyword.
- **Purpose.** Gives a one-line summary of the operation that the keyword performs.
- **Keyword type.** Identifies the keyword as a *statement* or as a *function.* (None of the keywords are operators.)
- **Execution options.** Indicates situations in which you can execute the keyword:
  - From the keyboard.
  - In CALC mode.
  - After THEN or ELSE in an IF ... THEN ... ELSE statement.
  - While the HP-71 is operating ▪▪ an HP-IL device (not ▬ controller). This is given only for HP-IL words.

- **Syntax diagram.** Defines the required and optional components within the statement or function for proper syntax. Parameters shown within brackets are optional. Parameters shown in a vertical stack are alternatives.
- **Examples.** Illustrates and explains some ways that the keyword can be used, and shows some possible syntax variations.
- **Input parameters.** Defines the parameters used in the syntax diagram, gives their default values (if applicable), and lists restrictions on parameter values or structure. (This heading isn't included for keywords that use no parameters.)
- **Operation.** Gives a detailed description of the keyword's operation and other information that's useful for learning and using the keyword.
- **Related keywords.** Lists other keywords that either influence the results of the subject keyword or else are similar in function.

DELETE#

Deletes one record from a text file.

■ Statement	■ Keyboard Execution
□ Function	□ CALC Mode
□ Operator	■ IF...THEN...ELSE

DELETE# *channel number , record number*

Example

```
DELETE# 5,14
```

Deletes the 14th record from the text file currently assigned to channel #5.

Input Parameters

Item	Description	Restrictions
channel number	Numeric expression rounded to an integer.	1 through 255.
record number	Numeric expression rounded to an integer.	

Operation

The DELETE# keyword deletes the specified record from the text file assigned to the specified channel number. Record numbers always begin at 0, so line number 1 is record number 0.

The channel number and the record number can be expressions. DELETE# rounds each of the resulting values to an integer.

DELETE# returns an error message if the assigned file is external, protected, or not a text file.

Related Keywords

ASSIGN#, INSERT#, REPLACE#, FILESZR

EDTEXT

Invokes the text editor.

■ Statement	■ Keyboard Execution
□ Function	□ CALC Mode
□ Operator	■ IF...THEN...ELSE

```
EDTEXT file specifier[, command string]
```

Examples

```
EDTEXT "SCREEN"
```
Runs the editor program, with "SCREEN" as the edit file.

```
EDTEXT "SCREEN", "L"
```
Runs the editor program, with "SCREEN" as the edit file. Begins by listing the file to the display device.

Input Parameters

Item	Description	Restrictions
file specifier	String expression or unquoted string.	File must be in RAM or IRAM.
command string	See description of editor command strings in section 3.	

Operation

The EDTEXT keyword starts the editor program. The optional command string permits you to have the editor begin immediate execution of editor commands that appear in the command string.

An error can cause the editor program to terminate without going through its normal exit path. If you are running the editor from another BASIC program, or from the FORTH or HP-41 environments, you can check for this situation by using DISP$ to read the display contents. If the result is other than Done: <filename>, then you will know that the editor has encountered a fatal error, the edit file may be in a corrupt state, and the editor key assignments may still be active. For example, from the FORTH environment, you can type the sequence

```
" EDTEXT "SCREEN"" BASICX " DISP$" BASICX DROP @ -102588 =
```

to edit the file "SCREEN". When the editor terminates, a true flag will be pushed on the stack if the editor terminated normally (here we are checking the numerical equivalent of the first three characters on the display to see if they match "Don", which translates to −102588).

Related Keywords

ASSIGN#, DELETE#, REPLACE#, FILESZR

FILESZR

Returns the number of records in a text file.

☐ Statement	■ Keyboard Execution
■ Function	☐ CALC Mode
☐ Operator	■ IF...THEN...ELSE

```
FILESZR (filename)
```

Example

```
X=FILESZR("SCREEN")
```

Sets the variable X equal to the number of records in the text file SCREEN.

Input Parameters

Item	Description	Restrictions
file name	String expression or quoted string.	Can not include a device specifier or CARD.

Operation

The FILESZR keyword returns the number of records in the file specified, if that file exists. If the file does not exist, or the operation fails for any other reason, a negative number is returned. The absolute value of the negative number is the error number of the error that caused the function to fail.

Related Keywords

INSERT#, DELETE#, REPLACE#

FORTH

Transfers HP-71 operation to the FORTH environment.

■ Statement	■ Keyboard Execution
□ Function	□ CALC Mode
□ Operator	□ IF...THEN...ELSE

```
FORTH
```

Operation

Keyboard execution of FORTH (it is not programmable) causes the HP-71 to exit the BASIC or HP-41 environments and transfer control to the FORTH environment. The message HP-71 FORTH 1A is displayed. Subsequent keyboard input is interpreted by the FORTH outer interpreter.

If the HP-71 is turned off while FORTH is active, it will automatically reenter the FORTH environment when the HP-71 is turned back on.

Execution of the FORTH word BYE will return the HP-71 to BASIC.

Because of the complete access to the HP-71 memory space provided by FORTH, it is quite possible for a FORTH program to store inappropriate data into HP-71 operating system RAM. In many cases, this will cause a memory lost condition. Following a memory loss, the HP-71 will return to the BASIC environment.

Related Keywords

FORTH$, FORTHF, FORTHI, FORTHX

FORTH$

Returns to ■ BASIC string variable the contents of ■ string defined in the FORTH environment by an address and character count on the FORTH data stack.

☐ Statement	■ Keyboard Execution
■ Function	☐ CALC Mode
☐ Operator	■ IF...THEN...ELSE

```
FORTH$
```

Examples

`A$=FORTH$`	Returns the value of the FORTH string to the BASIC variable `A$`.
`C$=C$&FORTH$`	Concatenates the FORTH string to `C$`.

Operation

`FORTH$` reads a string specified by the address and character count on the FORTH data stack and returns its value to a BASIC string variable. The contents of the FORTH data stack must already have been established prior to execution of `FORTH$`. If there are fewer than two values on the data stack when `FORTH$` is executed, an error will occur, producing the message `FTH ERR:empty stack`.

When `FORTH$` is executed, two values are dropped from the top of the FORTH data stack. There is no other effect on the FORTH environment. If the FTH41RAM file does not exist or is not positioned properly, the message `FTH ERR:FTH41RAM not in place` is displayed.

Related Keywords

`FORTH, FORTHF, FORTHI, FORTHX`

FORTHF

Returns the contents of the FORTH floating-point X-register to a BASIC numeric variable.

☐ Statement	■ Keyboard Execution
■ Function	■ CALC Mode
☐ Operator	■ IF...THEN...ELSE

```
FORTHF
```

Examples

```
X=FORTHF
```
Copies the contents of the FORTH X-register to the BASIC variable X.

```
X=SIN(FORTHF)
```
Computes the sine of the contents of the X-register and places the result in the BASIC variable X.

```
FORTHX'" A" BASICF FWORD'
B=FORTHF
```
Copies the BASIC variable A to the FORTH X-register, then executes a FORTH word FWORD, and returns the resulting value from the X-register to the BASIC variable B.

Operation

FORTHF allows floating-point numeric data in the FORTH environment to be accessed from the BASIC environment. FORTHF copies the contents of the FORTH floating X-register to a BASIC numeric variable. The contents of the FORTH floating-point stack remain unchanged, and there is no other effect on the FORTH environment.

The FORTH environment can be configured prior to execution of FORTHF through the keyword FORTHX. If the FTH41RAM file does not exist or is not positioned properly, the message FTH ERR:FTH41RAM not in place is displayed.

Related Keywords

FORTH, FORTH$, FORTHI, FORTHX

FORTHI

Returns the top value from the FORTH data stack to a BASIC numeric variable.

☐ Statement	■ Keyboard Execution
■ Function	■ CALC Mode
☐ Operator	■ IF...THEN...ELSE

```
FORTHI
```

Examples

```
I=FORTHI
```
Moves the top value from the FORTH data stack to the BASIC variable I.

```
I=FORTHI^2
```
Computes the square of the value on the FORTH data stack and places the result in the BASIC variable I.

```
FORTHX'" I" BASICI FWORD'
B=FORTHI
```
Copies the BASIC variable I to the FORTH data stack, then executes a FORTH word FWORD, and returns the resulting top value from the data stack to the BASIC variable B.

Operation

FORTHI allows values contained on the FORTH data stack to be accessed from the BASIC environment. FORTHI moves the value on the top of the FORTH data stack to a BASIC numeric variable. The value is dropped from the data stack, but there is no other effect on the FORTH environment.

If there are no values on the data stack when FORTHI is executed, an error will occur, producing the message FTH ERR:empty stack. The FORTH environment can be configured prior to execution of FORTHF through the keyword FORTHX. If the FTH41RAM file does not exist or is not positioned properly, the message FTH ERR:FTH41RAM not in place is displayed.

Related Keywords

FORTH, FORTH$, FORTHF, FORTHX

FORTHX

Executes a FORTH command string.

■ Statement	■ Keyboard Execution
□ Function	□ CALC Mode
□ Operator	■ IF...THEN...ELSE

```
FORTHX command string [,parameter list]
```

Example

```
FORTHX "DROP + . TYPE CR",
"Hello",1,2,3
```

Push onto the FORTH data stack the address and character count of the string "Hello," and the values 1, 2, and 3; then execute the FORTH words DROP, +, ., TYPE, and CR.

Input Parameters

Item	Description	Restrictions
command string	String expression.	Contains valid FORTH words.
parameter list	Numeric expressions and string expressions, separated by commas.	Maximum of 14 parameters.

Operation

The FORTHX keyword allows you to execute FORTH routines from the BASIC environment. The optional parameter list is a list of up to 14 string or numeric expressions, separated by commas. Each item in the list is pushed onto the FORTH data stack: numbers as single length numbers, and strings each as two numbers representing the address and character count of the string. After the parameters are placed on the stack, the sequence of FORTH words specified in the command string is executed, following which control is returned to the BASIC environment.

BASICX can not be included in the command list—the FORTH/BASIC interface does not permit reentrant execution.

The strings passed to FORTH in the parameter list are created in temporary memory. FORTH words can copy those strings to FORTH string variables, or concatenate them to existing strings, but you should not attempt to write other strings to the addresses of the temporary FORTHX strings.

Related Keywords

FORTH, FORTH$, FORTHF, FORTHI

HP41

Transfers HP-71 operation to the HP-41 environment.

■ Statement	■ Keyboard Execution
□ Function	□ CALC Mode
□ Operator	□ IF...THEN...ELSE

```
HP41
```

Operation

Keyboard execution of `HP41` (it is not programmable) causes the HP-71 to exit the BASIC operating system environment and transfer control to the HP-41 environment. The message `HP-41 Emulator 1A` is displayed. Subsequent keyboard input is interpreted by the FORTH outer interpreter.

If the HP-71 is turned off while FORTH is active, it will automatically reenter the HP-41 environment when the HP-71 is turned back on.

Executing `BASIC` returns the HP-71 to BASIC.

Related Keywords

`FORTH, FORTH$, FORTHF, FORTHI, FORTHX`

INSERT#

Inserts one record into a text file.

■ Statement	■ Keyboard Execution
□ Function	□ CALC Mode
□ Operator	■ IF...THEN...ELSE

```
INSERT# channel number , record number ; new record
```

Example

```
INSERT# 5,14;"Hello there"
```

Inserts the string "Hello there" into the file currently assigned to channel #5, as record 14. The former record 14 becomes record 15.

Input Parameters

Item	Description	Restrictions
channel number	Numeric expression rounded to an integer.	1 through 255.
record number	Numeric expression rounded to an integer.	
new record	String expression.	

Operation

The INSERT# keyword inserts the new record at the record number in the file assigned to the specified channel number. The new record is an HP-71 string expression. The channel number and the record number can be expressions. Record numbers always begin at 0, so line number 1 is record number 0. INSERT# rounds each of the resulting values to an integer.

The new record is inserted ahead of the record previously numbered at the record number. The former record, and all subsequent records, have their records numbers incremented incremented by 1.

INSERT# returns an error message if the assigned file is external, protected, or not a text file.

Related Keywords

ASSIGN#, DELETE#, REPLACE#, FILESZR

MSG$

Returns the message string corresponding to a specified error number.

☐ Statement	■ Keyboard Execution
■ Function	☐ CALC Mode
☐ Operator	■ IF...THEN...ELSE

```
MSG$(error number)
```

Example

```
A$=MSG$(58)
```

Places the message string associated with error #58 into the string variable A$.

Input Parameters

Item	Description	Restrictions
error number	Numeric expression.	Valid error number.

Operation

The MSG$ keyword provides access to the error message strings generated by the HP-71 operating system, the HP-41 Translator ROM, or any other LEX file. MSG$(*n*) returns the string corresponding to the *n*th error.

MSG$ is a generalization of the keyword ERRM$, which returns the message string associated with the most recent error.

Related Keywords

ERRN, ERRL, ERRM$

REPLACE#

Replaces one record in a text file.

■ Statement	■ Keyboard Execution
□ Function	□ CALC Mode
□ Operator	■ IF...THEN...ELSE

```
REPLACE# channel number , record number ; new record
```

Example

```
REPLACE# 5,14;"Hello there"
```

Replaces the 14th record in the text file currently assigned to channel #5, with the string "Hello there".

Input Parameters

Item	Description	Restrictions
channel number	Numeric expression rounded to an integer.	1 through 255.
record number	Numeric expression rounded to an integer.	
new record	String expression.	

Operation

The REPLACE# keyword replaces a specified record, in the text file assigned to the specified channel number, with a new record. The new record is an HP-71 string expression. The channel number and the record number can be expressions. Record numbers always begin at 0, so line number 1 is record number 0. REPLACE# rounds each of the resulting values to an integer.

REPLACE# returns an error message if the assigned file is external, protected, or not a text file.

Related Keywords

ASSIGN#, DELETE#, INSERT#, FILESZR

SCROLL

Scrolls the display to a position and waits for a key to be pressed.

■ Statement	■ Keyboard Execution
□ Function	□ CALC Mode
□ Operator	■ IF...THEN...ELSE

```
SCROLL position
```

Example

```
DISP "Hello there" @ SCROLL 4
```

Display the string "Hello there," with the fourth character in the string as the first character in the display, so that the display shows "lo there."

Input Parameters

Item	Description	Restrictions
position	Numeric expression rounded to an integer.	1 through 96.

Operation

The SCROLL keyword enables you to display a string, under program control, that can be scrolled from the keyboard. Execution of SCROLL causes the current display string to shift so that the character in the position specified by the numeric expression is the leftmost character in the display. Execution halts, so that a user can press the left- and right-arrow keys to scroll the display. Execution resumes when any other key is pressed (the pressed keycode is placed in the key buffer). The number input with SCROLL must be greater than zero.

SEARCH

Finds a string in a text file.

☐ Statement	■ Keyboard Execution
■ Function	☐ CALC Mode
☐ Operator	■ IF...THEN...ELSE

```
SEARCH(search string, column number, begin line, end line, channel)
```

Example

```
X=SEARCH("Hello",5,1,99,2)
```

Searches the file assigned to channel #2 for the string "Hello." The search starts in column 5, line 1, and extends through line 99.

Input Parameters

Item	Description	Restrictions
search string	String expression.	
column number	Numeric expression rounded to an integer.	1 through 9999
begin line	Numeric expression rounded to an integer.	0 through 9999
end line	Numeric expression rounded to an integer.	0 through 9999
channel	Numeric expression rounded to an integer.	1 through 255

Operation

The SEARCH keyword enables you to determine the location of a specified string within an HP-71 text file. If the search is successful, SEARCH returns a value in the format *nnn.cccllll*, where *nnn* is the record number, *ccc* is the column number, and *lll* is the length of the matched string. If the search is unsuccessful, zero is returned.

The search string can be any string expression, and the other parameters can be any numeric expression. Each input value is rounded to an integer. A zero is returned for an empty file.

Related Keywords

INSERT#, DELETE#, REPLACE#

Appendix D

FORTH Words

This appendix describes all FORTH words in the HP-41 Pac. The words appear in ASCII order. Each entry shows the word, its pronunciation, its use of the data stack, and a brief description of the word's operation. A word EXAMPLE might have the following entry:

EXAMPLE	*(Example)*	n_1 n_2 → n_3

Perform the specified operation on n_1 and n_2, replacing them on the data stack with the result n_3. (Before EXAMPLE is executed, n_2 is on the top of the stack. After EXAMPLE is executed, n_3 is on the top of the stack.)

Some descriptions categorize words as COMPILE, IMMEDIATE, FORTH, or HP-41. These indicate the following:

- COMPILE indicates that the word is intended for use only during compilation. Direct execution of the word can give meaningless or dangerous results; where appropriate, a FTH ERR: compile only error occurs.
- IMMEDIATE indicates that the word is executed, rather than compiled, when encountered during compilation.
- FORTH indicates that this word (or version of the word), is available in the FORTH vocabulary, but not in the HP41V vocabulary.
- HP-41 indicates that his word (or version of the word) is available only in the HP41V vocabulary, and not in the FORTH vocabulary.

If no categories are specified, the word works as described in both the FORTH and HP-41 environments.

Note: The HP-41 vocabulary contains many HP-41 functions that depend on special data structures existing only in the HP-41 environment. These functions are not considered proper FORTH words, and therefore are not included in this appendix. Refer to appendix E for a list of all HP-41 functions.

Notation

The stack-use diagrams use the following variables to represent various types of data.

Definition of Stack Variables

Variable	Type of Data
n	A signed (twos complement) 20-bit integer.
un	An unsigned 20-bit integer.
d	A signed (twos complement) 40-bit integer.
ud	An unsigned 40-bit integer.
flag	A signed (twos complement) 20-bit value, either −1 (true) or 0 (false).
c	A 20-bit value whose two low-order nibbles represent an ASCII character.
addr	A 20-bit address.
count	A 20-bit value whose two low-order nibbles represent the number of characters in a string.
str	A 40-bit value comprising *addr* and *count*. *Count* is on top and tells how many characters are to be found at *addr*.

Errors

Many FORTH words require one or more parameters on the data stack. When a word is executed with too few parameters on the stack, unpredictable errors will occur. The error message `FTH ERR: empty stack` might be displayed, but only after the operation is carried out on spurious parameters. These spurious parameters come from the terminal input buffer (TIB), which resides above the data stack. If a result is returned, it will be written into the TIB, and an error message like `FTH ERR: xYzgt not recognized` occurs when FORTH tries to interpret this result as a character string containing FORTH words and data.

FORTH is similar to assembly language in its lack of user protection. In most cases FORTH will attempt to perform a specified operation, even if the operation will cause a `Memory Lost` condition. For instance, it is easy to write a FORTH loop that pushes a value onto the data stack 1,000,000 times. Execution of this loop will overwrite the user dictionary, the FORTH system variables, and the BASIC O/S variables Eventually the machine will be too confused to continue and will perform a cold start. In other cases you might need to perform an `INIT 3` to recover normal HP-71 operation.

FORTH Glossary

!	*(Store)*	*n addr* →

Store *n* at *addr*.

"	*(Quote)*	→ *str*

Used in the form: " *ccc*"

IMMEDIATE. In execute mode: Take the characters *ccc*, terminated by the next ", from the input stream, and store them in a temporary string variable at the PAD. The string variable's header shows a maximum length of 80 characters or the current length, whichever is greater. Any other word that returns another temporary string will wipe out the first string.

In compile mode: Compile into the dictionary the runtime address of ", two bytes for the length of the string *ccc* (maximum length = current length), and the string itself. A string must be contained on a single line of a source file.

#	*(Sharp)*	ud_1 → ud_2

Used in the form: <# ### #>

Divide ud_1 by BASE, convert the remainder to an ASCII character, place this character in an output string, and return the quotient ud_2. Used in pictured output conversion; refer to <#.

#>	*(Sharp-greater)*	*ud* → *addr n*

End pictured output conversion. #> drops *ud* and returns the text address and character count. (These are suitable inputs for TYPE.)

#S	*(Sharp-s)*	*ud* → *0 0*

Convert *ud* into digits (as by repeated execution of #), adding each digit to the pictured numeric-output text until the remainder is zero. A single zero is added to the output if *ud* = 0. Used between <# and #>.

#TIB	*(Number-t-i-b)*	→ *addr*

Return the address of the variable #TIB, which contains the number of bytes in the terminal input buffer. Set by QUERY.

'	*(Tick)*	→ *addr*

Used in the form: ' *name*

Return the CFA of *name*.

'STREAM	*(Tick-stream)*	→ *addr*

Return the address of the next character in the input stream.

%	*(Percent)*	→

Replace x with $x*y/100$. The original value of x is saved in the LAST X register.

%CH	*(Percent change)*	→

Replace x with $100*(x-y)/y$. The original value of x is saved in the LAST X register.

(	*(Paren)*	→

Used in the form: (*ccc*)

IMMEDIATE. Consider the characters *ccc*, delimited by), as a comment to be ignored by the text interpreter. The blank following (is not part of *ccc*. (may be freely used while interpreting or compiling. A comment must be contained on a single line of a source file.

*	*(Times)*	n_1 n_2 → n_3

FORTH: Return the arithmetic product of n_1 and n_2.

*	*(Times)*	→

HP-41: Execute F*.

*/	*(Times-divide)*	n_1 n_2 n_3 → n_4

Multiply n_1 and n_2, divide the result by n_3, and return the quotient n_4. The product of n_1 and n_2 is maintained as an intermediate 40-bit value for greater precision in the division.

*/MOD	*(Times-divide-mod)*	n_1 n_2 n_3 → n_4 n_5

Multiply n_1 and n_2, divide the result by n_3, and return the remainder n_4 and the quotient n_5. The product of n_1 and n_2 is maintained as an intermediate 40-bit value for greater precision in the division.

+	*(Plus)*	n_1 n_2 → n_3

FORTH: Return the arithmetic sum of n_1 and n_2.

+	*(Plus)*	→

HP-41: Execute F+.

+!	*(Plus-store)*	*n addr* →

Add *n* to the 20-bit value at *addr*.

+BUF	*(Plus-Buff)*	$addr_1$ → $addr_2$ *flag*

Advance the mass-storage-buffer address ($addr_1$) to the address of the next buffer ($addr_2$). +BUF returns a false flag if $addr_2$ is the address of the buffer currently pointed to by PREV; otherwise, +BUF returns a true flag.

,	*(Comma)*	*n* →

Used in the form: `1234 ,`

Allot five nibbles and store *n* in the dictionary.

−	*(Minus)*	n_1 n_2 → n_3

FORTH: Subtract n_2 from n_1 and return the difference n_3.

−	*(Minus)*	→

HP-41: Execute F−.

−TRAILING	*Dash-trailing*	*addr count¶ →addr countæ*

Adjust the character count of the text beginning at *addr* to exclude trailing blanks.

.	*(Dot)*	*n* →

Convert *n* according to `BASE` and display the result in a free-field format with one trailing blank. Display a minus sign if *n* is negative.

."	*(Dot-quote)*	→

Used in the form: `." ccc"`

COMPILE, IMMEDIATE: Compile the characters *ccc*, delimited by ", so that later execution will transmit *ccc* to the current display device. The blank following ." is not part of *ccc*. A string must be contained of ■ single line of a source file.

.(	*(Dot-paren)*	→

Used in the form: .(ccc)

IMMEDIATE: Display the characters *ccc*, delimited by). The blank following .(is not part of *ccc*. A string must be contained on a single line of a source file.

.S	*(Dot-S)*	→

Print the contents of the stack as unsigned integers, starting with the top of the stack. .S doesn't alter the stack.

/	*(Divide)*	n_1 n_2 → n_3

FORTH: Divide n_1 by n_2, and return the quotient n_3. Division by 0 always yields 0.

/	*(Divide)*	→

HP-41: Execute F/.

/MOD	*(Divide-mod)*	n_1 n_2 → n_3 n_4

Divide n_1 by n_2, and return the remainder n_3 and quotient n_4.

0	*(Zero)*	→ *0*

FORTH: Return the constant 0.

0	*(Zero)*	→

HP-41: Lift the floating-point stack, place 0 into the X-register, and set number entry flag 22.

0<	*(Zero-less)*	*n* → *flag*

Return a true flag if $n < 0$; otherwise, return a false flag.

1/X	*(Reciprocal-of-X)*	→

Divide 1.0 by the contents of the X-register. `1/X` places the result in the X-register and the original value of x in the LAST X register.

10^X	*(10-to-the-X)*	→

Raise 10 to the power contained in the X-register. `10^X` places the result in the X-register and the original value of x in the LAST X register.

2	*(Two)*	→ 2

FORTH: Return the constant 2.

2	*(Two)*	→

HP-41: Lift the floating-point stack, place 2 into the X-register, and set number entry flag 22.

2*	*(Two-times)*	n → $2n$

Return the product of n and 2.

2+	*(Two-plus)*	n → $n+2$

Increment n by 2.

2–	*(Two-minus)*	n → $n-2$

Decrement n by 2.

2/	*(Two-divide)*	n → $n/2$

Divide n by 2 and return the result. `2/` produces $n/2$ by shifting n one bit to the right and extending the sign bit.

2DROP	*(Two-drop)*	d →

Drop the double number (or two single numbers) on the top of the data stack.

2DUP	*(Two-dup)*	d_1 → d_1 d_1

Duplicate the double number (or pair of single numbers) on the top of the data stack.

2OVER	*(Two-over)*	d_1 d_2 → d_1 d_2 d_1

Make a copy of the second double number (or third and fourth single numbers) on the data stack.

2SWAP	*(Two-swap)*	d_1 d_2 → d_2 d_1

Reverse the order of the two double numbers on the top of data stack.

3	*(Three)*	→ 3

FORTH: Return the constant 3.

3	*(Three)*	→

HP-41: Lift the floating-point stack, place 3 into the X-register, and set number entry flag 22.

4N@	*(Four-n-fetch)*	*addr* → *n*

Return the four-nibble (two-byte) quantity located at *addr*.

5+	*(Five-plus)*	n → $n+5$

Increment n by 5.

5–	*(Five-minus)*	*n* → *n*−5

Decrement *n* by 5.

:	*(Colon)*	→

Used in the form: : *name* . . . ;

Create a word definition for *name* in the compilation vocabulary and set compilation state. The search order is changed so that the first vocabulary in the search order is replaced by the compilation vocabulary. The compilation vocabulary is unchanged. The text from the input stream is subsequently compiled. *name* is called a *colon definition.* The newly created word definition for *name* cannot be found in the dictionary until the corresponding ; is successfully processed.

;	*(Semicolon)*	→

Used in the form: : *name* . . . ;

IMMEDIATE, COMPILE. Stop compilation of a colon definition. ; compiles EXIT into the dictionary, clears the smudge bit (so that this colon definition can be found in the dictionary), and sets execute state.

<	*(Less-than)*	n_1 n_2 → *flag*

Return a true flag if $n_1 < n_2$; otherwise, return a false flag.

<#	*(Less-sharp)*	→

Initialize pictured numeric output. The words <#, #, #S, HOLD, SIGN, and #> can specify the conversion of a double number into an ASCII-character string stored in right-to-left order.

<>	*(Not-equal)*	n_1 n_2 → *flag*

Return a true flag if $n_1 \neq n_2$; otherwise, return a false flag.

=	*(Equals)*	n_1 n_2 → *flag*

Return a true flag if $n_1 = n_2$; otherwise, return a false flag.

>	*(Greater-than)*	n_1 n_2 → *flag*

Return a true flag if $n_1 > n_2$; otherwise, return a false flag.

>BODY	*(To-body)*	$addr_1$ → $addr_2$

Return the PFA ($addr_2$) of the word whose CFA is $addr_1$. ($addr_2 = addr_1 + 5$.)

>IN	*(To-in)*	→ *addr*

Return the address of the variable >IN, which contains the current offset within the input stream. The offset is expressed in nibbles and points to the first position past the first blank.

>R	*(To-R)*	*n* →

COMPILE. Transfer *n* to the return stack.

?	*(Question-mark)*	*addr* →

Used in the form: `HEX 2FCC5 ?`

Display the number at *addr* using the current `BASE` and the . (dot) format.

?COMP	*(Query-comp)*	→

COMPILE. Issue a `FTH ERR: compile only` message if not in compile mode.

?DUP	*(Query-dup)*	*n* → *n* *(n)*

Duplicate *n* if $n \neq 0$.

?STACK	*(Query-stack)*	→

Issue a `FTH ERR: empty stack` message if the stack pointer is above the bottom of the stack; or issue a `FTH ERR: full stack` message if the stack pointer has grown into the pad.

?TERMINAL	*(Query-terminal)*	→ *flag*

Return a true flag if a key has been pressed and placed in the key buffer; otherwise, return a false flag.

@	*(Fetch)*	*addr* → *n*

Return the number stored at *addr*.

ABORT	*(Abort)*	→

Reset the data and return stacks, close all files, set execution mode, set `FORTH` as the current and context vocabulary, and return control to the terminal.

ABORT"	*(Abort-quote)*	*flag* →

Used in the form: `: name ... ABORT" ccc" ... ;`

COMPILE, IMMEDIATE. If *flag* is true, display the character string *ccc* (delimited by ") and execute `ABORT`; otherwise, drop the flag and continue execution. The character string must be contained on a single line of a source file.

ABS	*(Absolute)*	*n* → |*n*|

FORTH: Return the absolute value of *n*.

ABS	*(Absolute)*	→

HP-41: Execute FABS.

ACOS *(A-cos)* →

Calculate the arc cosine of the contents of the X-register, according to the currently active angular mode. ACOS places the result in the X-register and the original value of x in the LAST X register.

ADJUSTF *(Adjust-f)* *addr n → flag*

Adjust a file by n nibbles, starting at *addr* and moving toward greater addresses, and return a true flag if successful or a false flag if not. ADJUSTF enlarges the file for positive n or shrinks the file for negative n.

ALLOT *(Allot)* *n →*

Add n bytes to the parameter field of the most recently defined word (regardless of the CURRENT and CONTEXT vocabularies).

AND *(And)* n_1 n_2 → n_3

Return the bit-by-bit AND of n_1 and n_2.

ASC *(Ascii)* *str → n*

Return the ASCII value of the first character in the string specified by *str*.

ASIN *(A-sine)* →

Calculate the arc sine of the contents of the X-register, according to the currently active angular mode. ASIN places the result in the X-register and the original value of x in the LAST X register.

ATAN *(A-tan)* →

Calculate the arc tangent of the contents of the X-register, according to the currently active angular mode. ATAN places the result in the X-register and the original value of x in the LAST X register.

BASE *(Base)* → *addr*

Return the address of the variable BASE, which contains the current numeric-conversion base.

BASIC *(Basic)* →

Exit the FORTH or HP-41 environments, and enter the BASIC environment.

BASIC$ *(Basic-dollar)* str_1 → str_2

Used in the form:
```
" A$" BASIC$
" A$[1,3]" BASIC$
```

Return the current value of a BASIC string expression (specified by str_1) to the pad as a FORTH string (specified by str_2.)

BASICF *(Basic-f)* *str* →

Used in the form:
```
" A" BASICF
" A1/A6" BASICF
" A9*PI" BASICF
" TIME" BASICF
```

Return the current value of a BASIC numeric expression (specified by *str*) to the FORTH X-register, lifting the floating-point stack.

BASICI *(Basic-i)* *str* → *n*

Used in the form:
```
" A" BASICI
" A1/A6" BASICI
```

Return the current value of a BASIC numeric expression (specified by *str*). An overflow error occurs if the variable's value exceeds FFFFF.

BASICX *(Basic-x)* *str* →

Used in the form:
```
" RUN 'JOE'" BASICX
" BEEP" BASICX
" A=PI" BASICX
" 10 DISP A" BASICX
```

Pass a string (specified by *str*) to the BASIC system for parsing and editing/execution, and then return to FORTH.

BEEP *(Beep)* →

HP-41: Sound the HP-41 4-tone beep.

BEGIN . . . UNTIL →

Used in the form: . . . BEGIN *actions flag* UNTIL . . .

IMMEDIATE, COMPILE: Execute *actions* and test *flag*; if *flag* is false, repeat; if *flag* is true, skip to the word following UNTIL.

BEGIN . . . WHILE . . . REPEAT →

Used in the form: . . . BEGIN *actions*$_1$ *flag* WHILE *actions*$_2$ REPEAT . . .

IMMEDIATE, COMPILE: Execute *actions*$_1$ and test *flag*; if flag is true, execute *actions*$_2$ and repeat; if *flag* is false, skip to the word following REPEAT.

BL *(Blank)* → *c*

Return 32_{10}, the ASCII value for a space or blank.

BLK *(B-l-k)* → *addr*

Return the address of the variable BLK, which contains the number of the line being interpreted from the active file. The value of BLK is an unsigned number; if it is zero, the input stream is taken from the keyboard device.

BLOCK *(Block)* *n* → *addr*

Return the address of the first byte in the mass-storage-buffer copy of line *n* in the active file. If line *n* hasn't already been copied from the file (in RAM or on mass storage) into a mass storage buffer, BLOCK does so.

BYE *(Bye)* →

Exit the FORTH or HP-41 environment and return control to the BASIC environent.

C!	*(C-store)*	*n addr* →

Store the two low-order nibbles of *n* at *addr*.

C,	*(C-comma)*	*n* →

ALLOT one byte and store the two low-order nibbles of *n* at HERE.

C@	*(C-fetch)*	*addr* → *byte*

Return the contents of the byte at *addr*. The three high-order nibbles of the five-nibble stack entry are 0.

C@+	*(C-at-plus)*	str_1 → str_2 *c*

Return *c*, the first character in the string specified by str_1, and str_2, where $addr_2 = addr_1 + 2$ and $count_2 = count_1 - 1$. If $count_1 = 0$, $c = 0$ and $str_2 = str1$.

CASE . . . OF . . . ENDOF . . . ENDCASE	*(Case Statements)*	*n* →

Used in the form:

```
. . . CASE
n1 OF actions1 ENDOF actions1'
n2 OF actions2 ENDOF actions2'
n3 OF actions3 ENDOF actions3'
. . .
ENDCASE . . .
```

IMMEDIATE, COMPILE. Starting with the first case statement ($i = 1$):

- If $n = n_i$, drop *n*, execute $actions_i$, and skip to the word following ENDCASE.
- If $n \neq n_i$, execute $actions_i'$ and examine the next case statement. (If there are no more case statements, drop *n* and skip to the word following ENDCASE). Note that each optional $actions_i'$ can alter the value of *n* (the number on the top of the stack) tested by the next case statement.

CHIRP	→

Sound the HP-71 error beep.

CHR$ *(Char-dollar)* *n* → *str*

Convert the two low-order nibbles of *n* into an ASCII character and place it in a string specified by *str*. The string is a temporary string of length 1, located on the pad.

CHS *(Change-sign)* →

Replace *x*, the contents of the X-register, with $-x$.

CLKEYS *(Clear keys)* →

Purge the current HP-71 keys file. (CLKEYS uses BASICX).

CLOCK *(Clock)* →

Place the current HP-71 clock time into the X-register, lifting the floating-point stack.

CLST *(Clear-stack)* →

Replace *x*, *y*, *z*, and *t* (the contents of the X-, Y-, Z-, and T-registers) with 0.

CLX *(Clear x)* →

Replace *x*, the contents of the X-register, with 0.

CLOSEALL *(Close-all)* →

Close all open files (that is, files with an open FIB entry).

CLOSEF *(Close-f)* *n* →

Close the file whose FIB# is *n*.

CMOVE *(C-move)* $addr_1$ $addr_2$ un →

Move un bytes, first moving the byte at $addr_1$ to $addr_2$ and finally moving the byte at $addr_1 + 2(un - 1)$ to $addr_2 + 2(un - 1)$. If $un = 0$, nothing is moved.

CMOVE> *(C-move-up)* $addr_1$ $addr_2$ un →

Move un bytes, first moving the byte at $addr_1 + 2(un - 1)$ to $addr_2 + 2(un - 1)$ and finally moving the byte at $addr_1$ to $addr_2$. If $un = 0$, nothing is moved.

COMPILE *(Compile)* →

Used in the form: : *name*$_1$ . . . COMPILE *name*$_2$ . . .

COMPILE. Compile the CFA of *name*$_2$ when *name*$_1$ is executed. Typically *name*$_1$ is an immediate word and *name*$_2$ is not; COMPILE ensures that *name*$_2$ is compiled, not executed, when *name*$_1$ is encountered in a new definition.

CONBF *(Con-buff)* n_1 n_2 → *flag*

Contract by n_1 nibbles the general-purpose buffer whose ID# is n_2, and return a true flag; or return a false flag if such a buffer doesn't exist. If the specified buffer contains fewer than n_1 nibbles, CONBF contracts it to 0 nibbles. n_1 must not exceed FFF.

CONSTANT *(Constant)* n →

Used in the form: n CONSTANT *name*

Create a dictionary entry for *name*, placing n in its parameter field. Later execution of *name* will return n.

CONTEXT *(Context)* → *addr*

Return the address of the variable CONTEXT, which specifies which vocabulary to search first during interpretation of the input stream. (Word searches through successive parent vocabularies are discussed in section 2.)

CONVERT (Convert) d_1 $addr_1$ → d_2 $addr_2$

Accumulate the string of digits beginning at $addr_1 + 2$ into the double number d_1, and return the result d_2 and the address $addr_2$ of the next non-digit character. For each character that is a valid digit in BASE, CONVERT converts the digit into a number, multiplies the current double number (initially d_1) by BASE, and adds the converted digit to the current double number. When CONVERT encounters a non-digit character, it returns the current double number and the non-digit character's address.

COS (Cos) →

Calculate the cosine of the contents of the X-register, according to the currently active angular mode. COS places the result in the X-register and the original value of x in the LAST X register.

COUNT (Count) $addr_1$ → $addr_2$ n

Return the address ($addr_2$) of the first character, and the character count (n), of the counted string beginning at $addr_1$. The first byte at $addr_1$ must contain the character count n. The following diagram shows the parameters for a three-character text string:

	Address	Contents	
$addr_1$ →	1000	3	← n
$addr_2$ →	1002	A	
	1004	B	
	1006	C	

CR (C-r) →

Send a carriage-return and line-feed to the current display device.

CREATE (Create) →

Used in the form: CREATE *name*

Create a standard dictionary entry for *name* without allotting any parameter-field memory. Later execution of *name* will return *name*'s PFA. Words that use CREATE directly are called *defining words*.

CREATEF	*(Create-f)*	*str n* → *addr* *str n* → false

Create a text file in RAM whose name is specified by *str* and that contains *n* nibbles. If successful, CREATEF returns the address of the beginning of the file header (which contains the file name); otherwise, it returns a false flag. If the specified string exceeds eight characters, the file name will be the first eight characters.

CRLF	*(C-r-l-f)*	→ *str*

Return *str* specifying the two-character string constant containing the ASCII characters carriage-return and line-feed. This string can be concatenated with other strings for use with words such as OUTPUT.

CURRENT	*(Current)*	→ *addr*

Return the address of the variable CURRENT, which specifies the vocabulary to receive new word definitions.

D+	*(D-plus)*	d_1 d_2 → d_3

Return the arithmetic sum of d_1 and d_2.

D−	*(D-minus)*	d_1 d_2 → d_3

Subtract d_2 from d_1 and return the difference d_3.

D-R	*(Degrees-to-radians)*	→

Replace x with $\pi x/180$. The original value of x is saved in the LAST X register.

D.	*(D-dot)*	*d* →

Display *d* according to BASE in a free-field format, with a leading minus sign if *d* is negative.

D.R	*(D-dot-R)*	*d n* →

Display *d* (according to BASE) right-justified in a field *n* characters wide.

D<	*(D-less-than)*	d_1 d_2 → *flag*

Return a true flag if $d_1 < d_2$; return a false flag otherwise.

DABS	*(D-abs)*	d_1 → $\lvert d \rvert$

Return the absolute value of *d*.

DEC	*(To decimal)*	→

Convert *x* as an octal number to decimal form, where *x* is an integer in the range $-7777777777777 \leq x \leq +777777777777$, containing no digits 8 or 9. The original value of *x* is saved in the LAST X register.

DECIMAL	*(Decimal)*	→

Set the input-output numeric conversion BASE to ten.

DEFINITIONS	*(Definitions)*	→

Set the CURRENT vocabulary to match the CONTEXT vocabulary.

DEG	*(Degrees)*	→

HP-41: Set degrees trigonometric mode. Clear HP-41 user flags 42 and 43.

DEGREES	*(Degrees)*	→

Select degrees trigonometric mode.

DEPTH	*(Depth)*	→ *n*

Return *n*, the number of items on the data stack (not counting *n* itself).

DIGIT	*(Digit)*	*c* n_1 → n_2 true *c* n_1 → false

If *c* is a valid digit in base n_1, return that digit's binary value (n_2) and a true flag; otherwise, return a false flag.

DLITERAL	*(D-literal)*	*d* →

COMPILE, IMMEDIATE. Compile *d* into the word being defined, such that *d* will be returned when the word is executed.

DNEGATE	*(D-negate)*	*d* → −*d*

Return the twos complement of a double number *d*.

DO . . . +LOOP	*(Do, Plus-loop)*	n_1 n_2 →

Used in the form: . . . DO *actions* *n* +LOOP . . .

COMPILE, IMMEDIATE. Execute a definite loop, each time incrementing the loop index by *n*. DO moves n_1 (the loop limit) and n_2 (the initial value of the loop index) to the return stack, with n_2 on top, and then executes *actions*. +LOOP increments the index by *n* (which can be negative) and repeats *actions*, until the index is incremented across the boundary between *n* − 1 and *n*. For example,

```
10 1 DO actions 1 +LOOP
```

will execute *actions* nine times, with values of the index from 1 through 9; and

```
-10 -1 DO actions -1 +LOOP
```

will execute *actions* ten times, with values of the index from −1 through −10. DO . . . +LOOP may be nested within control structures.

DO . . . LOOP

n_1 n_2 →

Used in the form: . . . DO *actions* LOOP . . .

COMPILE, IMMEDIATE. Execute a definite loop, each time incrementing the loop index by 1. DO moves n_1 (the loop limit) and n_2 (the initial value of the loop index) to the return stack, with n_2 on top, and then executes *actions*. LOOP increments the index by 1 and repeats *actions*, until the index is incremented from $n - 1$ to n. DO . . . LOOP may be nested within control structures.

DOES>

(Does)

Used in the form: : *name* . . . CREATE . . . DOES> . . . ;

COMPILE, IMMEDIATE. Define the run-time action of a word created by a defining word. DOES> marks the termination of the defining part of the defining word *name* and begins the definition of the run-time action for words that will later be defined by *name*.

DROP

(Drop) *n* →

Drop the top number from the stack.

DSIZE

(D-size) *n* →

Makes *n* nibbles available for definitions in the user dictionary.

DUP

(Dup) *n* → *n* *n*

Return a second copy of the top number on the stack.

EMIT

(Emit) *c* →

Transmit the character *c* to the current display device.

ENCLOSE	*(Enclose)*	*addr c → addr* n_1 n_2 n_3

Examine the string that begins at *addr*, and return:

- n_1, the nibble offset from *addr* to the first character that doesn't match the delimiter character *c*.
- n_2, the nibble offset from *addr* to the first delimiter character *c* that follows non-delimiter characters in the string.
- n_3, the nibble offset from *addr* to the first unexamined character.

An ASCII null is treated as an unconditional delimiter.

END$	*(End-dollar)*	str_1 *n →* str_2

Create a temporary string (specified by str_2) consisting of the *n*th character and all subsequent characters in the string specified by str_1. (RIGHT$ is similar but takes substring length, not character position, for a parameter.)

ENG	*(Engineering)*	*n →*

Select engineering display mode with $n + 1$ significant digits displayed, for $0 \leq n \leq 11$.

ENG	*(Engineering)*	*→*

HP-41, IMMEDIATE: Used in form ENG *n*. Select engineering display mode with $n+1$ significant digits displayed, for $0 \leq n \leq 11$. Affects HP-41 display and digit flags (user flags 36 through 41).

ENTER	*(Enter)*	*addr* n_1 *→ addr* n_2 *addr* n_1 *c 0 → addr* n_2

FORTH: Receive up to n_1 bytes of data from the HP-IL device whose address is specified by PRIMARY and SECONDARY, and store the data at *addr* and above (greater addresses). ENTER leaves *addr* on the stack and returns n_2, the actual number of characters received. Executing ENTER requires the HP 82401A HP-IL Interface.

There are two options for termination in addition to the limit of n_1 characters:

- If system flag −23 is set, ENTER will terminate when an End Of Transmission message is received.
- If the argument on the top of the stack is 0, ENTER interprets the second argument on the stack to be a character and will terminate when an incoming character matches this character. This option is effective only when system flag −23 is clear.

ENTER^ *(Enter)* →

HP-41: Execute FENTER.

EOF *(E-o-f)* → *flag*

Return a true flag if there are no more records in the active file; otherwise, return a false flag. EOF examines the record length of the next record in the file specified by the FIB# in SCRFIB. It assumes that the current pointer into the file is pointing at the next record length and that the file is a text file.

EXECUTE *(Execute)* *addr* →

Execute the dictionary entry whose CFA is on the stack.

EXIT *(Exit)* →

COMPILE. Terminate execution. Don't use EXIT within a DO loop.

EXPBF *(Expand-buff)* n_1 n_2 → *flag*

Expand by n_1 nibbles the general-purpose buffer whose ID# is n_2, and return a true flag; or return a false flag if such a buffer doesn't exist, if the resulting size would exceed 2K bytes, if there is insufficient memory, or if n_1 is negative. n_1 must not exceed FFF.

EXPECT96 *(Expect-96)* *addr* →

Accept 96 characters from the keyboard (or fewer characters followed by [ENDLINE]), append two null bytes, and store the result at *addr* and above (greater addresses). EXPECT96 also copies the text into the Command Stack.

E^X *(E-to-the-x)* →

Raise e to the power contained in the X-register. E^X places the result in the X-register and the original value of x in the LAST X register.

E^X−1 *(E-to-the (x−1))* →

Raise *e* to the power computed by subtracting 1 from the contents of the X-register.

F* *(F-times)* →

Multiply the contents of the X- and Y-registers. F* drops the stack (duplicating T into Z), then places the result in the X-register and the original value of *x* in the LAST X register.

F+ *(F-plus)* →

Add the contents of the X- and Y-registers. F+ drops the stack (duplicating T into Z), then places the result in the X-register and the original value of *x* in the LAST X register.

F− *(F-minus)* →

Subtract the contents of the X-register from the contents of the Y-register. F− drops the stack (duplicating T into Z), then places the result in the X-register and the original value of *x* in the LAST X register.

F. *(F-dot)* →

Display the contents of the X-register according to the currently active display format. F. doesn't alter the contents of the X-register.

F/ *(F-divide)* →

Divide the contents of the Y-register by the contents of the X-register. F/ drops the stack (duplicating T into Z), then places the result in the X-register and the original value of *x* in the LAST X register.

FABS *(F-abs)* →

Take the absolute value of the contents of the X-register. FABS places the result in the X-register and the original value of *x* in the LAST X register.

FACT *(Factorial)* →

Replace x with $x!$ (x must be an integer). The original value of x is saved in the LAST X register.

FCONSTANT *(F-constant)* →

Used in the form: *floating-point number* FCONSTANT *name*

Create a dictionary entry for *name*. When *name* is later executed, the value that was in the X-register when *name* was created is placed in the X-register, lifting the floating-point stack.

FDROP *(F-drop)* →

Copy the contents of the Y-register into the X-register, the contents of the Z-register into the Y-register, and the contents of the T-register into the Z-register. The previous contents of the X-register are lost.

FENCE *(Fence)* → *addr*

Return the address of the variable FENCE, which contains the address below which the dictionary is protected from FORGET.

FENTER *(F-enter)* →

Copy the contents of the Z-register into the T-register, the contents of the Y-register into the Z-register, and the contents of the X-register into the Y-register. The previous contents of the T-register are lost.

FILL *(Fill)* *addr un byte* →

Fill memory from *addr* through *addr* + (2*un* − 1) with *un* copies of *byte*. FILL has no effect if $un = 0$.

FIND *(Find)* $addr_1$ → $addr_2$ n

Search the dictionary (in the currently active search order) for the word contained in the counted string at $addr_1$. If the word is found, FIND returns the word's CFA (= $addr_2$) and either $n = 1$ (if the word is immediate) or $n = -1$ (if the word isn't immediate). If the word isn't found, FIND returns $addr_2 = addr_1$ and $n = 0$.

FINDBF	*(Find-buff)*	*n* → *addr* *n* → false

Return the start-of-data address in the general-purpose buffer whose ID# is *n*, or return a false flag if such a buffer doesn't exist.

FINDF	*(Find-f)*	*str* → *addr* *str* → false

Search main RAM for the file whose name is specified by *str*, and return either the address of the beginning of the file header (if successful) or a false flag (if not). If the specified string exceeds eight characters, FINDF considers only the first eight characters.

FIRST	*(First)*	→ *addr*

Return the address of the variable FIRST, which contains the address of the first (lowest addressed) mass storage buffer in the FORTHRAM file.

FIX	*(Fix)*	*n* →

FORTH: Select fixed-point display mode with *n* decimal places, $0 \leq n \leq 11$.

FIX	*(Fix)*	→

HP-41, IMMEDIATE: Used in form FIX *n* to select fixed point display mode. Affects HP-41 display and digits flags (user flags 36 through 41).

FLITERAL	*(F-literal)*	→

IMMEDIATE, COMPILE. Compile the value *x* (the contents of the X-register) into the dictionary. When the colon definition is later executed, *x* will be placed in the X-register, lifting the floating-point stack.

FLUSH	*(Flush)*	→

Unassign all mass storage buffers.

FORGET *(Forget)* →

Used in the form: FORGET *name*

Delete from the dictionary *name* (which must be in the search order that begins with the CURRENT vocabulary) and all words added to the dictionary after *name* (regardless of their vocabulary). Failure to find *name* in the search order that begins with the CURRENT vocabulary is an error condition.

FORTH *(Forth)* →

FORTH, IMMEDIATE: Set the CONTEXT vocabulary to FORTH, the name of the first vocabulary in RAM. Because all vocabularies ultimately chain to the FORTH vocabulary, the word FORTH can be found regardless of the CONTEXT vocabulary.

FORTH *(Forth)* →

HP-41: Exit the HP-41 environment to the FORTH environment. Restore the default NUMBER and INTERPRET, clear the HP-41 active variable, and make FORTH the CONTEXT vocabulary.

FP *(F-p)* →

Take the fractional part of the contents of the X-register. FP places the result in the X-register and the original value of x in the LAST X register.

FRC *(Frac)* →

HP-41: Alternate spelling for FP, above.

FSTR$ *(F-string-dollar)* → *str*

Create a string (specified by *str*) that represents the contents of the X-register.

FTOI *(F-to-i)* → *n*

Convert x (the contents of the X-register) to an integer and return it to the data stack. If $|x| >$ FFFFF, an overflow error occurs. FTOI takes the absolute value of x, rounds it to the nearest integer, and converts it to a five-nibble value. If x was positive, FTOI returns this result; if x was negative, FTOI returns the twos complement of this result.

FVARIABLE *(F-variable)* →

Used in the form: `FVARIABLE` *name*

Create a dictionary entry for *name*, and allocate eight bytes for its parameter field. Subsequent execution of *name* will return *name*'s PFA. This parameter field will hold the contents of the variable, which must be initialized by the application that creates it.

GRAD *(Grads)* →

Set grads trigonometric mode (set HP-41 user flag 42 and clear flag 43).

GROW *(Grow)* *n* → *flag*

Enlarge the user dictionary by *n* nibbles and return a true flag; or if there is insufficient memory, return a false flag (without enlarging the dictionary).

H. *(H-dot)* *un* →

Display *un* in base 16 as an unsigned number with one trailing blank.

HERE *(Here)* → *addr*

Return the address of the next available dictionary location.

HEX *(Hex)* →

Set `BASE` to sixteen.

HMS *(Hours-minutes-seconds)* →

Convert *x* from decimal hours to hours-minutes-seconds (hh.mmssss) format.

HMS+	*(Hours-minutes-seconds plus)*	→

Replace x with $x+y$, where x, y, and $x+y$ are in hours-minutes-seconds (hh.mmssss) format, dropping the floating-point stack. x is saved in the LAST X register.

HMS−	*(Hours-minutes-seconds minus)*	→

Replace x with $y-x$, where x, y, and $y-x$ are in hours-minutes-seconds (hh.mmssss) format, dropping the floating-point stack. x is saved in the LAST X register.

HOLD	*(Hold)*	*c* →

Insert character c into a pictured numeric output string. Used between <# and #>.

HP41	*(HP-41)*	→

IMMEDIATE: Enter the HP-41 environment. Select HP41V as the context and current vocabularies, replace the default NUMBER and INTERPRET with HP-41 versions, create HP-41 data registers word R41 (if necessary, set decimal mode, initialize HP-41 user flags, set math exception traps to HP-41 default values.

HP41V	*(HP-41V)*	→

Set the context vocabulary to HP41V.

HR	*(Hours)*	→

Convert x from hh:mmssss (hours-minutes-seconds) format to decimal hours.

I	*(I)*	→ *n*

Used in the form: `... DO ... I ... LOOP ...`

FORTH, COMPILE, IMMEDIATE: Return the current value of the DO-loop index.

I	*(I)*	→

HP-41: Place the value of the BASIC variable I into the X-register, lifting the floating-point stack.

IF . . . THEN	*flag* →

Used in the form: . . . IF *actions* THEN . . .

COMPILE, IMMEDIATE. Execute *actions* if and only if *flag* is true. IF . . . THEN conditionals may be nested.

IF . . . THEN . . . ELSE	*flag* →

Used in the form: . . . IF *actions*$_1$ ELSE *actions*$_2$ THEN . . .

COMPILE, IMMEDIATE. Execute *actions*$_1$ if and only if *flag* is true; execute *actions*$_2$ if and only if *flag* is false. IF . . . ELSE . . . THEN conditionals may be nested within control structures.

IMMEDIATE	*(Immediate)*	→

Mark the most recent dictionary entry as a word to be executed, not compiled, when encountered during compilation.

INT	*(Int)*	→

HP-41: Alternate spelling for IP, below.

INTERPRET	*(Interpret)*	→

Interpret the input stream to its end, beginning at the offset contained in >IN. The input stream comes from the TIB (if BLK contains 0) or from the mass storage buffer containing the *n*th line of the active file (if BLK contains *n*.)

IP	*(I-p)*	→

Take the integer part of the contents of the X-register. IP places the result in the X-register and the original value of *x* in the LAST X register.

ITOF	*(I-to-f)*	*n* →

Convert *n* into a floating-point number and place it in the X-register, lifting the floating-point stack.

J	*(J)*	→ *n*

Used in the form: `... DO ... DO ... J ... LOOP ... LOOP ...`

FORTH, COMPILE, IMMEDIATE. Return the index of the next outer loop. Used within nested `DO ... LOOP` structures.

J	*(J)*	→

HP-41: Place the value of the BASIC variable J into the X-register, lifting the floating-point stack.

KEY	*(Key)*	→ *c*

Return the low-order seven bits of the ASCII value of the next key pressed. If the key buffer is empty, wait for a key to be pressed.

KILLBF	*(Kill-buff)*	*n* → *flag*

Delete the general-purpose buffer whose ID# is *n*, and return a true flag; or return a false flag if no such buffer exists.

L	*(L)*	→ *addr*

FORTH: Return the address of the floating-point LAST X register.

L	*(L)*	→

HP-41: Place the value of the BASIC variable L into the X-register, lifting the floating-point stack.

LASTX	*(Last-x)*	→

Lift the floating-point stack and copy the contents of the LAST X register into the X-register.

LATEST	*(Latest)*	→ *addr*

Return the NFA of the most recent word in the CURRENT vocabulary.

LEAVE	*(Leave)*	→

COMPILE, IMMEDIATE: Skip to the word after the next LOOP or +LOOP. LEAVE terminates the loop and discards the control parameters. Used only within a DO . . . LOOP or +LOOP construct.

LEFT$	*(Left-dollar)*	str_1 *n* → str_2

Create a temporary string (specified by str_2) consisting of the first *n* characters in the string specified by str_1.

LGT	*(Log-ten)*	→

Calculate the common log (base 10) of the contents of the X-register. LGT places the result in the X-register and the original value of *x* in the LAST X register.

LIMIT	*(Limit)*	→ *addr*

Return the address of the variable LIMIT, which contains the first address beyond the mass-storage-buffer area.

LINE#	*(Line-number)*	→ *addr*

Return the address of the variable LINE#, which contains the number of the line being loaded from the active file (specified by SCRFIB).

LITERAL	*(Literal)*	*n* →

COMPILE, IMMEDIATE: Compile *n* into the word being defined, such that *n* will be returned when the word is executed.

LN	*(Natural log)*	→

Calculate the natural log (base e) of the contents of the X-register. LN places the result in the X-register and the original value of x in the LAST X register.

LN1+X	→

Replace x with the natural log of $(1+x)$. The original value of x is saved in the LAST X register.

LOAD	*(Load)*	→

HP-41, IMMEDIATE: Used in the form LOAD *file name*. Compile an HP-41 program contained in the intermediate text file named *file name*. Not programmable.

LOADF	*(Load-f)*	*str* →

Interpret the entire file specified by *str*. If the file cannot be opened for any reason (doesn't exist, wrong type, already opened, etc.), LOADF will give the error message FTH ERR: *filename* cannot load.

LOG	*(Log)*	→

HP-41: Alternate spelling for LGT, above.

M*	*(Mixed-multiply)*	n_1 n_2 → d

Return the double-number product d of two single numbers n_1 and n_2. All numbers are signed.

M/	*(Mixed-divide)*	d n_1 → n_2 n_3

Divide the double number d by the single number n_1, and return the single-number remainder n_2 and the single-number quotient n_3. All numbers are signed.

M/MOD	*(Mixed-divide-mod)*	ud_1 un_1 → un_2 ud_2

Divide the double number ud_1 by the single number un_1, and return the single-number remainder un_2 and the double-number quotient ud_2. All numbers are unsigned.

MAKEBF	*(Make-buff)*	*n* → *addr ID#* true *n* → false

Create a buffer *n* nibbles long and return a true flag, the buffer ID#, and the address of the beginning of data area in the buffer; or if unsuccessful (not enough memory, no free buffer ID#s), return a false flag. *n* cannot exceed 4095_{10}.

MAX	*(Max)*	n_1 n_2 → n_3

Return the greater of n_1 and n_2.

MAXLEN	*(Max-length)*	*str* → *n*

Return the maximum length (that is, bytes of memory allotted in the dictionary) for the string specified by *str*.

MIN	*(Min)*	n_1 n_2 → n_3

Return the smaller of n_1 and n_2.

MOD	*(Mod)*	n_1 n_2 → n_3

FORTH: Divide n_1 by n_2, and return the remainder n_3 with the same sign as n_1.

MOD	*(Mod)*	→

HP-41: Replace *x* with *y* MOD(*x*), dropping the floating-point stack.

N@	*(N-fetch)*	*addr* → *n*

Return the contents of the nibble at *addr*. The four high-order nibbles of *n* are zeros.

N!	*(N-store)*	*n addr* →

Store at *addr* the low-order nibble of *n*.

NALLOT	*(N-allot)*	*n* →

Add *n* nibbles to the parameter field of the most recently defined word (regardless of the CURRENT and CONTEXT vocabularies).

NEGATE	*(Negate)*	*n* → *−n*

Return the twos complement of *n*.

NFILL	*(N-fill)*	*addr un n* →

Fill memory from *addr* through *addr* + (*un* − 1) with *un* copies of the low-order nibble in *n*. NFILL has no effect if *un* = 0.

NMOVE	*(N-move)*	$addr_1$ $addr_2$ *un* →

Move *un* nibbles, first moving the nibble at $addr_1$ to $addr_2$ and finally moving the nibble at $addr_1$ + (*un* − 1) to $addr_2$ + (*un* − 1). NMOVE has no effect if *un* = 0.

NMOVE>	*(N-move-up)*	$addr_1$ $addr_2$ *un* →

Move *un* nibbles, first moving the nibble at $addr_1$ + (*un* − 1) to $addr_2$ + (*un* − 1) and finally moving the nibble at $addr_1$ to $addr_2$. NMOVE> has no effect if *un* = 0.

NOP	*(No-op)*	→

"No operation" FORTH secondary (: NOP ;).

NOT	*(Not)*	n_1 → n_2

Return the ones complement (true Boolean NOT) of n_1.

NULL$	*(Null-dollar)*	→ *str*

Create a temporary string (specified by *str*) in the pad, with maximum length = 80 and current length = 0.

NUMBER	*(Number)*	*addr* → *d* *addr* →

Examine the counted string at *addr* and convert it into a double number *d*.

- If the string contains a decimal point, NUMBER tries to convert it into a floating-point number and place it in the X-register, lifting the floating-point stack. If the string contains a decimal point but is not a legal floating-point number, a Data Type error occurs.
- If the string does not contain a decimal point, NUMBER tries to convert it into an integer number and return it to the data stack. If the string isn't a legal integer, a FTH ERR: NUMBER not recognized error occurs.

OCT	*(To octal)*	→

Convert x from a decimal integer to octal digits. x must be an integer in the range $-68719476735 \leq x \leq +68719476735$. The original value of x is saved in the LAST X register.

OKFLG	*(Okay-flag)*	→ *addr*

Return the address of the variable OKFLG. If the value of OKFLG is 0, the OK ⟨*n*⟩ message is shown when the FORTH system is ready for input; otherwise, the message is suppressed.

ON	*(On)*	→

HP-41: Set continuous on mode. Set HP-41 flag 44 and system flag −3.

ONERR *(On-error)* → *addr*

Return the address of the variable ONERR, which contains the CFA of the user's error routine. The value of ONERR is checked when a FORTH-system error occurs. If the value of ONERR is zero, the error is processed by the system's error routine. If the value of ONERR is not zero, control is transferred instead to the user's error routine. The stacks are not reset. The BASIC keywords FORTH and FORTHX set the value of ONERR to zero.

OPENF *(Open-f)* *str* → *t*
str → *str f*

Open an FIB for the file whose name is specified by *str*, and store the FIB# into SCRFIB. If successful, OPENF returns a true flag. If the file was empty or there was a problem in opening the file, OPENF returns *str* and a false flag.

OR *(Or)* n_1 n_2 → n_3

Return the bit-by-bit inclusive OR of n_1 and n_2.

OUTPUT *(Output)* *addr* *n* →

Send *n* bytes, stored at *addr* through *addr* + 2(*n* − 1), to the HP-IL device whose address is specified by PRIMARY and SECONDARY. Executing OUTPUT requires the HP 82401A HP-IL Interface.

OVER *(Over)* n_1 n_2 → n_1 n_2 n_1

Return a copy of the second number on the stack.

P-R *(Polar-to-rectangular)* →

Replace the contents of the X- and Y-registers (x = radius, y = angle) with the equivalent vector (x,y) expressed in rectangular coordinates, according to the current angular mode. The original value of x is saved in the LAST X register.

PAD *(Pad)* → *addr*

Return the address of the pad, which is a scratch area used to hold character strings for intermediate processing.

PI	*(Pi)*	→

Lift the floating-point stack, and place a 12-digit representation of π into the X-register.

PICK	*(Pick)*	n_1 → n_2

Return a copy of the n_1-th entry on the data stack (not counting n_1 itself). For example, `1 PICK` is equivalent to `DUP`, and `2 PICK` is equivalent to `OVER`.

POS	*(Pos)*	str_1 str_2 → n str_1 str_2 → false

Search the string specified by str_2 for a substring that matches the string specified by str_1, and return the position of the first character in the matching substring (or a false flag if there is no matching substring).

PREV	*(Prev)*	→ *addr*

Return the address of the variable PREV, which contains the address of the most recently referenced mass storage buffer.

PRIMARY	*(Primary)*	→ *addr*

Return the address of the variable PRIMARY, which specifies an HP-IL address. The valid range for PRIMARY is 0 through 31, and the default value is 1. (The contents of PRIMARY and SECONDARY specify which HP-IL device to use with `ENTER` and `OUTPUT`. If system flag −22 is clear, the contents of PRIMARY alone specify a simple address; if system flag −22 is set, the contents of PRIMARY and SECONDARY specify an extended address.)

QUERY	*(Query)*	→

Accept characters from the current keyboard until 96 characters are received or an (END LINE) character is encountered, and store them in the TIB. `QUERY` sets `#TIB` to the value of `SPAN`.

QUIT	*(Quit)*	→

Clear the return stack, set execution mode, and return control to the keyboard. No message is displayed.

R-D *(Radians-to-degrees)* →

Replace x with $180*x/\pi$. The original value of x is saved in the LAST register.

R-P *(Rectangular-to-polar)* →

Replace the contents (x,y) of the X- and Y-registers with the equivalent vector (x = radius, y = angle) expressed in polar coordinates, according to the current angular mode. The original value of x is saved in the LAST X register.

R> *(R-from)* → *n*

COMPILE: Remove *n* from the top of the return stack and return a copy to the data stack.

R@ *(R-fetch)* → *n*

COMPILE: Return a copy of the number on the top of the return stack.

RAD *(Radians)* →

HP-41: Set radians trigonometric mode. Set user flag 43 and clear flag 42.

RADIANS *(Radians)* →

Select RADIANS angular mode.

RCL *(Recall)* *addr* →

FORTH: Lift the floating-point stack and place in the X-register the floating-point number found at *addr*.

RDN	*(Roll-down)*	→

Roll down the floating-point stack. RDN copies from the T-register into the Z-register, from the Z-register into the Y-register, from the Y-register into the X-register, and from the X-register into the T-register.

RIGHT$	*(Right-dollar)*	str_1 n → str_2

Create a temporary string (specified by str_2) consisting of the last (rightmost) n characters in the string specified by str_1. (END$ is similar but takes character position, not substring length, for a parameter.)

ROLL	*(Roll)*	n →

Move the nth entry on the data stack (not counting n itself) to the top of the stack. For example, 2 ROLL is equivalent to SWAP, and 3 ROLL is equivalent to ROT.

ROOM	*(Room)*	→ n

Return the current number of nibbles available for words in the dictionary.

ROT	*(Rote)*	n_1 n_2 n_3 → n_2 n_3 n_1

Rotate the top three entries on the data stack, bringing the deepest to the top of the stack.

RP!	*(R-p-store)*	→

Reset the return stack to 0 addresses.

RP@	*(R-p-fetch)*	→ *addr*

Return the current value of the return-stack pointer.

RP0	*(R-p-zero)*	→ *addr*

Return the address of the system variable RP0, which contains the address of the bottom of the return stack. (The bottom of the return stack has a greater address than the top.)

RUP	*(Roll-Up)*	→

Roll up the floating-point stack. RUP copies from the X-register into the Y-register, from the Y-register into the Z-register, from the Z-register into the T-register, and from the T-register into the X-register.

R^	*Roll-up)*	→

HP-41: Alternate spelling for RUP, below.

S!	*(S-store)*	str_1 str_2 →

Store the contents of the string specified by str_1 into the string specified by str_2.

S−>D	*(Sign-extend)*	*n* → *d*

Return a signed double number *d* with the same value and sign as the signed single number *n*.

S0	*(S-zero)*	→ *addr*

Return the address of the bottom of the data stack.

S<	*(S-less)*	str_1 str_2 → *flag*

Return a true flag if the string specified by str_1 is "less than" the string specified by str_2, or a false flag if not. S< first compares the ASCII values of the first characters; if they are equal, it then compares the second characters, and so on. ABC is defined to be less than ABCD.

S<&	*(S-left-concatenate)*	str_1 str_2 → str_3

Append the contents of the string specified by str_2 to the end of the string specified by str_1, and return str_3, the address and length of the resulting string. The address of str_3 is the address of str_1; the length of str_3 is the combined length of str_1 and str_2. If the concatenation would exceed str_1's maximum length, no concatenation occurs and $str_3 = str_1$. Either str_1 or str_2 can specify a temporary string in the pad. The < sign indicates that the left string will contain the result of the concatenation.

S=	*(S-equals)*	str_1 str_2 → *flag*

Return a true flag if the two strings are equal, or a false flag if not. S= compares only the current length and contents of the strings, not the maximum length or old contents stored beyond current length.

S>&	*(S-right-concatenate)*	str_1 str_2 → str_3

Append the contents of the string specified by str_2 to the end of the string specified by str_1, and return str_3, the address and length of the resulting string. The address of str_3 is the address of str_2; the length of str_3 is the combined length of str_1 and str_2. If the concatenation would exceed str_2's maximum length, no concatenation occurs and $str_3 = str_2$. Either str_1 or str_2 can specify a temporary string in the pad. The > sign indicates that the right string will contain the result of the concatenation.

SCI	*(Scientific)*	*n* →

FORTH: Select scientific display mode with $n + 1$ significant digits displayed, $0 \leq n \leq 11$.

SCI	*(Scientific)*	→

HP-41, IMMEDIATE: Used in form SCI *n*. Affects HP-41 display and digits flags (user flags 36 through 41).

SCRFIB	*(Screen-f-i-b)*	→ *addr*

Return the address of the variable SCRFIB, which contains the FIB# of the currently active file (or 0 if no file is being loaded).

SECONDARY	*(Secondary)*	→ *addr*

Return the address of the variable SECONDARY, which specifies the extended portion of an HP-IL address. The valid range for SECONDARY is from 0 through 31, and the default value is 0. (The contents of PRIMARY and SECONDARY specify which HP-IL device to use with ENTER and OUTPUT. If system flag −22 is clear, the contents of PRIMARY specify a simple address; if system flag −22 is set, the contents of PRIMARY and SECONDARY specify an extended address.)

SHRINK *(Shrink)* *n* → *flag*

Shrink the user's dictionary space (and consequently the FTH41RAM file) by *n* nibbles, and return ■ true flag; or return a false flag if there are fewer than *n* free nibbles in the dictionary.

SIGN *(Sign)* *n* →

FORTH: Insert the ASCII minus sign - into the pictured numeric output string if *n* is negative. Used between <# and #>.

SIGN *Sign)* →

HP-41: Replace *x* with +1 if *x* is positive or zero, −1 if *x* is negative, 0 if *x* is alpha data, NaN if *x* is NaN. The original value of *x* is saved in the LAST X register.

SIN *(Sine)* →

Calculate the sine of the contents of the X-register, according to the currently active angular mode. SIN places the result in the X-register and the original value of *x* in the LAST X register.

SMOVE *(S-move)* *str addr* →

Store at *addr* and above (greater addresses) the characters in the string specified by *str*.

SMUDGE *(Smudge)* →

Toggle the smudge bit in the latest definition's name field.

SP! *(S-p-store)* →

Reset the data stack to 0 items.

SP0 *(S-p-zero)* → *addr*

Return the address of the system variable SP0, which contains the address of the bottom of the data stack. (The address of the bottom of the data stack is greater than the address of the top.)

SP@	*(S-P-fetch)*	→ *addr*

Return *addr*, the address of the top of the data stack before SP@ was executed.

SPACE	*(Space)*	→

Transmit an ASCII space to the current display device.

SPACES	*(Spaces)*	*n* →

Transmit *n* spaces to the current display device. Take no action for $n \leq 0$.

SPAN	*(Span)*	→ *addr*

Return the address of the variable SPAN, which contains the count of characters actually read by the last execution of EXPECT96.

SQRT	*(Square-root)*	→

Calculate the square root of the contents of the X-register. SQRT places the result in the X-register and the original value of x in the LAST X register.

ST.	*(Stack dot)*	→

HP-41: Display the contents of the floating-point stack, in the format T Z Y X | L.

STATE	*(State)*	→ *addr*

Return the address of the variable STATE, which contains a non-zero value if compilation is occurring (or zero if not).

STD *(Standard)* →

Select the BASIC standard display format.

STO *(Store)* *addr* →

Store the contents of the X-register at *addr*.

STR$ *(String-dollar)* *d* → *str*

Convert the number *d* into a temporary string in the pad, specified by *str*.

STRING *(String)* *n* →

Used in the form: *n* STRING *name*.

Create a dictionary entry for *name*, allotting one byte for a maximum-length field (value = *n*), one byte for a current-length field (value = 0), and *n* bytes for the string characters.

STRING-ARRAY *(String-array)* n_1 n_2 →

Used in the form: n_1 n_2 STRING-ARRAY *name*

Create a dictionary entry for *name*, allotting one byte for the maximum-length field (value = n_1), one byte for the dimension field (value = n_2), and (n_1 + 2) bytes each for n_2 string-array elements. STRING-ARRAY fills in the maximum-length (value = n_1) and current-length (value = 0) fields for each string-array element.

Later execution of *n name* will return str_n, the address and current length of the *n*th element of the string array.

SUB$ *(Sub-dollar)* str_1 n_1 n_2 → str_2

Create a temporary string (specified by str_2) consisting of the n_1th through n_2th characters in the string specified by str_1.

SWAP	*(Swap)*	n_1 n_2 → n_2 n_1

Exchange the top two entries on the data stack.

SYNTAXF	*(Syntax-f)*	*str* → *flag*

Return a true flag if the string specified by *str* is a valid HP-71 file name, or return a false flag if not. If the specified string exceeds eight characters, SYNTAXF checks only the first eight characters.

T	*(T)*	→ *addr*

FORTH: Return the address of the floating-point T-register.

T	*(T)*	→

HP-41: Return the value of the BASIC variable T to the X-register, raising the floating-point stack.

TAN	*(Tan)*	→

Calculate the tangent of the contents of the X-register, according to the currently active angular mode. TAN places the result in the X-register and the original value of x in the LAST X register.

TIB	*(T-i-b)*	→ *addr*

Return the address of the terminal input buffer. The terminal input buffer can hold up to 96 characters.

TIME	*(Time)*	→

HP-41: Lift the floating point stack, and place the current reading of the HP-71 system clock, in H.MMSS format, into the X-register. Also, display the time in hh:mm:ss format (unless an HP-41 program is running).

TOGGLE *(Toggle)* *addr* n_1 →

Replace n_2 (the contents at *addr*) with the bit-by-bit logical value of (n_1 XOR n_2).

TRAVERSE *(Traverse)* $addr_1$ *n* → $addr_2$

Return the address of the opposite end (length byte or last character) of a definition's name field.

- If $n = 1$, $addr_1$ is the address of the length byte, and $addr_2$ is address of the last character.
- If $n = -1$, $addr_1$ is the address of the last character, and $addr_2$ is the address of the length byte.
- If n doesn't equal 1 or -1, $addr_1 = addr_2$.

TYPE *(Type)* *addr* *n* →

Transmit *n* characters, found at *addr* through *addr* + (2*n* − 1), to the current display device. TYPE transmits no characters for $n \leq 0$.

U. *(U-dot)* *un* →

Display *un* (according to BASE) as an unsigned number in a free-field format with one trailing blank.

U< *(U-less-than)* un_1 un_2 → *flag*

Return a true flag if $un_1 < un_2$, or return a false flag if not.

UM* *(U-m-times)* un_1 un_2 → *ud*

Return the double-number product *ud* of two single numbers un_1 and un_2. All numbers are unsigned.

UM/MOD *(U-m-divide-mod)* ud_1 un_1 → un_2 un_3

Divide the double number ud_1 by the single number un_1, and return the single-number remainder un_2 and the single-number quotient un_3. All numbers are unsigned.

USE	*(Use)*	→ *addr*

Return the address of the variable USE, which contains the address of the next mass storage buffer available for use.

VAL	*(Val)*	*str* → *d* *str* →

Convert the string specified by *str* into a number.

- If the string contains a decimal point, VAL tries to convert it into a floating-point number and place it in the X-register, lifting the floating-point stack. If the string contains a decimal point but is not a legal floating-point number, a `Data Type` error occurs.
- If the string does not contain a decimal point, VAL tries to convert it into an integer number and return it to the data stack. If the string is not a legal integer, a `FTH ERR: VAL not recognized` error occurs.

VARIABLE	*(Variable)*	→

Used in the form: `VARIABLE` *name*

Create a dictionary entry for *name*, allotting five nibbles for its parameter field. Later execution of *name* will return *name*'s PFA. This parameter field will hold the contents of the variable, which must be initialized by the application that created it.

VOCABULARY	*(Vocabulary)*	→

Used in the form: `VOCABULARY` *name*

Create (in the `CURRENT` vocabulary) a dictionary entry for *name* that begins a new linked list of dictionary entries. Later execution of *name* will select *name* as the `CONTEXT` vocabulary. (Vocabularies are discussed in section 2.)

WARN	*(Warn)*	→ *addr*

Return the address of the variable WARN. If WARN contains a non-zero value, compiling a new word whose name matches an existing word causes a *name* `isn't unique` message to be displayed; if WARN contains 0, the message is suppressed.

WIDTH	*(Width)*	→ *addr*

Return the address of the variable WIDTH, which determines the maximum allowable length for the name of a word. The valid range for WIDTH is from 1 through 31.

WORD	*(Word)*	*c* → *addr*

Receive characters from the input stream until the non-zero delimiting character *c* is encountered or the input stream is exhausted, and store the characters in a counted string at *addr*. WORD ignores leading delimiters. If the input stream is exhausted as WORD is called, a zero-length string results.

X	*(X)*	→ *addr*

FORTH: Return the address of the floating-point X-register.

X	*(X)*	→

HP-41: Return the value of the BASIC variable X to the X-register, raising the floating-point stack.

X<>Y	*(X-exchange-y)*	→

Exchange the contents of the X- and Y-registers.

X#Y?	**X<=Y?**		→ *flag*
X<Y?	**X=0?**		
X=Y?	**X>=Y?**		
X≠Y?	**X>0?**	Floating-point Comparisons	
X≠0?	**X<=0?**		
X>0?	**X#0?**		
X>=0?			

Compare the contents of the X- and Y-registers, and return a true flag if the test is true or a false flag if not. The tests don't alter the contents of the X- and Y-registers.

XOR	*(X-or)*	n_1 n_2 → n_3

Return the bit-by-bit exclusive OR of n_1 and n_2.

XSIZE	*(X-size)*	→

Make x nibbles available for definitions in the user dictionary, where x id obtained from the X-register.

X^2	*(X-squared)*	→

Calculate the square of the contents of the X-register. X^2 places the result in the X-register and the original value of x in the LAST X register.

Y	*(Y)*	→ *addr*

FORTH: Return the address of the floating-point Y-register.

Y	*(Y)*	→

HP-41: Return the value of the BASIC variable Y to the X-register, raising the floating-point stack.

Y^X	*(Y-to-the-x)*	→

Raise the contents of the Y-register to the power contained in the X-register. Y^X places the result in the X-register and the original value of x in the LAST X register.

Z	*(Z)*	→ *addr*

FORTH: Return the address of the floating-point Z-register.

Z	*(Z)*	→

Return the value of the BASIC variable T to the X-register, raising the floating-point stack.

[*(Left-bracket)* →

IMMEDIATE: Suspend compilation. Subsequent text from the input stream will be executed.

['] *(Bracket-tick)* →

Used in the form: : *name*$_1$ . . . ['] *name*$_2$ . . .

COMPILE, IMMEDIATE: Compile the CFA of *name*$_2$ as a literal. An error occurs if *name*$_2$ is not found in the currently active search order. Later execution of *name*$_1$ will return *name*$_2$'s CFA.

[COMPILE] *(Bracket-compile)* →

Used in the form: . . . [COMPILE] *name* . . .

IMMEDIATE, COMPILE: Compile *name*, even if *name* is an IMMEDIATE word.

] *(Right-bracket)* →

Resume compilation. Subsequent text from the input stream is compiled.

Appendix E

Summary of the HP-41 Emulator Features

This appendix describes the HP-41 functions and capabilities that are included in the HP-41 Translator Pac, plus the general and specific differences in operation and results that you can expect between the emulator and the HP-41 itself.

HP-41 Functions

The basic function set included in the HP-41 emulator is the entire set of programmable functions contained in the HP-41C/CV calculator. In addition, the emulator includes certain functions from the HP-41CX and the HP 82160A HP-IL Module, listed below. In general, any HP-41 program that uses only the functions listed here can be executed by the emulator, with numerical results and alpha displays essentially identical to those obtained with the HP-41.

The following list of functions details the match between the HP-41 emulator and the HP-41CX. The function categories are taken from the Function Tables in the HP-41CX Owner's Manual Volume II.

- **System/Format Functions**. All functions included except: CLK12, CLK24, CLKT, CLKTD, DMY, MDY, PASN.
- **Clearing Functions.** Function included: CLD, CLKEYS, CLP, CLRG, CLRGX, CLΣ, CLST, CLX. Functions omitted: CLALMA, CLALMX, CLFL, CLRALMS, DELCHR, DELREC, PCLPS, PURFL, DEL.
- **Stack/Data Register Functions.** All functions included.
- **Numeric Functions.** All functions included.
- **Extended Memory Functions.** No functions. In effect, extended program memory is provided by the HP-71 file system.
- **Time Functions.** TIME is included.
- **Editing Functions.** Functions included: ON, OFF, SIZE, PSIZE, CAT, and CLP. All HP-41 program editing is done through the HP-71 Editor. ASN is replaced by the normal HP-71 key assignment method. PACK is performed automatically. SST and BST are available only in program editing.
- **Functions that Direct Program Execution.** All functions included, except CLOCK and GETP. The comparison functions X>=Y? and X>=0? are present in the emulator, but not in the HP-41CX.
- **Alpha Functions.** Functions included: ALENG, ANUM, AOFF, AON, ARCL, AROT, ASHF, ASTO, ATOX, AVIEW, CLA, POSA, PROMPT, XTOA. Functions omitted: ADATE, ATIME, ATIME24, ARCLREC, GETREC.
- **Interactive Functions**. Functions included: ADV, BEEP, PROMPT, PSE, TONE. Functions omitted: GETKEY, GETKEYX.
- **Printer Functions.** The following functions are included from the HP-41 HP-IL Module, which are not present in the HP-41CX: ACA, ACCHR, ACX, PRA, PRBUF, PRFLAGS, PRREG, PRREGX, PRΣ, PRSTK, PRX, SKPCHR. Printer control with flags 21 and 55 is the same as with the HP-41. The printer control provided by HP-41 flags 12 (double wide), 13 (lowercase), and 15 and 16 (printer mode) is not implemented in the emulator.

General Differences Between the HP-41 and the Emulator

Although the emulator usually duplicates the operation of the HP-41 in programs and calculations, there are several general differences between the HP-41 and the HP-41 emulator that can affect results. The most obvious differences are in the keyboards and user interface, as described in Section 2. You must consider the absence of HP-41 stack lift disable when you perform keyboard arithmetic, but programs written with HP-41 stack lift disable functions operate identically on the HP-71 and the HP-41 (with the single exception of the ANUM function; refer to page 170).

In addition, there are some subtle differences. The most pervasive of these is the different numerical accuracies of the HP-71 and the HP-41. Specifically, the HP-71 represents numbers with a 12-digit mantissa, and 3-digit exponent (dynamic range between 10^{-500} and 10^{500}), compared to the 10-digit mantissa and 2-digit exponent (range 10^{-100} to 10^{100}) of the HP-41. Internally, the HP-71 performs calculations with a 16-digit mantissa and 5-digit exponent, compared to the 14-digit mantissa and 3-digit exponent used by the HP-41. In general, calculations performed with the HP-41 emulator will be more accurate than the same operations performed on the HP-41. In some circumstances, this difference in numerical accuracy can produce dramatically different results in programs. For example, a program that causes a range error on the HP-41 with a result greater than 9.999999999E99 produces no error when run on the HP-71 unless the result exceeds 9.99999999999E499. Many HP-41 game programs use pseudo-random number routines that give different results when translated to the HP-71 because of the extra two mantissa digits provided by HP-71.

Mathematical Exceptions

The HP-71 provides a more sophisticated capability for handling special mathematical errors than the HP-41. The HP-71 mathematical exception treatment is preserved in the HP-41 emulator floating-point functions. From the BASIC environment, you can set the various exception traps to specify the system response to exceptions. However, some of these traps are altered when you enter the HP-41 environment.

When you type HP41 to enter the HP-41 environment, the INX and UNF traps are each set to value 1 to suppress the associated error messages that have no HP-41 counterpart. (You can choose to suppress warning messages, which do not halt program or function execution, by setting HP-71 system flag −1 in the BASIC environment.) The OVF (overflow) and IVL (invalid operation) traps are set to value 0 to produce errors analogous to the HP-41 DATA ERROR and OUT OF RANGE errors.

The state of the OVF (overflow) trap is controlled by HP-41 flag 24 (range error ignore), which is cleared initially. When flag 24 is cleared, this trap is set to value 0, which causes errors when the overflow exceptions occur. When flag 24 is set, the trap is set to value 1, which returns the MAXREAL value (±9.99999999999E499) for functions that cause an overflow, with a warning message if flag −1 is clear.

The DVZ (division-by-zero) trap is initially set to 0, so that functions causing this exception return an error. In the HP-41 emulator, this includes actual division by 0, LOG(0), LN(0), and TAN(90) (degrees). This is not strictly consistent with the HP-41, since TAN(90°) on the HP-41 is not an error.

You can override the emulator default trap settings from within the HP-41 environment by using the function BASICX to execute the BASIC trap functions. For example, to set the DVZ trap to value 1, type:

```
" TRAP(DVZ,1)" BASICX [END LINE]
```

When you set the DVZ trap to 1, division by 0, LOG(0), LN(0), and TAN(90°) return MAXREAL.

Extended Register and Numeric Label Range

The HP-41 emulator is capable of addressing 10,000 data registers (limited by available memory), numbered 0 through 9999. All of the two-part emulator functions that include a register number accept a 4-digit number—you are not limited to register numbers 0 through 99 for direct addressing. This does not affect programs translated from the HP-41, but you can use the full range of registers in programs you write on the HP-71 in HP-41 user language. (The functions CLREGX and PRREGX are exceptions; they are restricted to registers 0 through 999.)

Similarly, you can use numeric labels in the range LBL 0 through LBL 9999.

Trigonometric Modes

The HP-71 operating system does not provide a grads trigonometric mode, nor the corresponding annunciator. The HP-41/FORTH grads mode is provided through extensions of the trigonometric functions, but it is not a "global" mode in the same sense as the degrees and radians modes. Grads mode is determined within the HP-41/FORTH environment, as it is on the HP-41, by HP-41 flag 42. Grads mode is active when flag 42 is set, and inactive when flag 42 is clear. Since the HP-41 trigonometric mode is matched to the current HP-71 mode when you enter the HP-41/FORTH environment, you must reset grads mode any time you leave and then re-enter the HP-41/FORTH environment.

Display Formatting—Flags 28 and 29

The HP-71 does not provide any radix option or number digits separators. Therefore, the formatting effects of flags 28 and 29, particularly with respect to the alpha strings produced by ARCL, are not reproduced in the HP-41 emulator.

Automatic Execution—Flag 11

The HP-41 automatic execution feature, controlled by flag 11, is available in the HP-41 emulator. However, to obtain automatic program execution when you turn the HP-71 on, you must turn the calculator off using the OFF function executed from the keyboard or by a program.

Function-Specific Differences

The following is a list of all HP-41 emulator functions. Where appropriate, a description is included of how a function may work differently from the corresponding HP-41 function, other than the general differences that are described previously. If a particular function is listed only by name, its operation is the same as desribed in the HP-41 owner's manual.

Table E-1. HP-41 Functions Included in the HP-41 Emulator

+	Plus.
−	Minus.
*	Multiplied by.
/	Divided by.
1/X	Reciprocal.
10^X	Common exponential.

Table E-1. HP-41 Functions Included in the HP-41 Emulator (Continued)

ABS	Absolute value.
ACA	Accumulate alpha into print buffer.
ACCHR	Accumulate character specified in X.
ACX	Accumulate X in print buffer.
ACOS	Arc cosine.
ADV	Advance. Because of the variety of printers that can be connected to the HP-71, ADV will not print the contents of the print buffer right justified. Its action will be the same as PRBUF.
ALENG	Alpha length.
ANUM	Alpha number. ANUM is the only emulator function that cannot be guaranteed to reproduce the stack-lift behavior of its HP-41 counterpart. This is because the stack effect of ANUM is indeterminate. When the alpha register contains a number, ANUM raises the stack; otherwise, it has no effect at all. The translation program TRANS41 can not predict the stack behavior of ANUM. If there is a number in the alpha register, ANUM works the same as it does on the HP-41. If there is no number in the alpha register, ANUM performs a stack roll down (RDN). Furthermore, ANUM only recognizes numbers in HP-71 format. That is, digit separators are not allowed, and the period is the only acceptable radix.
AOFF	Alpha mode off.
AON	Alpha mode on.
ARCL	Alpha recall.
AROT	Alpha rotate.
ASHF	Alpha shift.
ASIN	Arc sine.
ASTO	Alpha store.
ATAN	Arc tangent.
ATOX	Alpha to X.
AVIEW	Alpha view.
BEEP	Beeper.
CAT	Catalog. The CAT function is similar to the HP-41 user program catalog function CAT 1. CAT lists the alpha labels currently in emulator memory, in last-to-first order (reversed from the HP-41). Program END's are not listed. Each label is displayed for approximately 1 second; if you press any key (don't use [ATTN]) during a label display, the program pointer will be positioned at that label. If you let CAT run uninterrupted, the program pointer will not move. The message No HP-41 programs indicates that no programs are currently loaded in emulator memory.
CF	Clear flag.
CHS	Change sign.
CLA	Clear alpha.
CLD	Clear display.
CLKEYS	Clear user key assignments. If there is no keys file, CLKEYS will return the warning WRN: File Not Found.

Table E-1. HP-41 Functions Included in the HP-41 Emulator (Continued)

CLP	Clear program. This function performs the function of the HP-41 CLP, but has an effect more like the function PCLPS. That is, CLP clears not only the program containing the specified alpha label, but also all programs (and any FORTH words) compiled after the specified program. The correct syntax is CLP"*label name*". The final quote is required only if there are any spaces in the label name, or if additional functions follow the CLP command in the input command line.
CLRG	Clear registers.
CLRGX	Clear registers by X. Data Error if |x|>999.
CLS	Clear summations.
CLST	Clear stack.
CLX	Clear X. CLX executed at the keyboard is equivalent to typing 0. It enters 0 into the X-register, lifting the stack. A subsequent number will not overwrite the 0, but instead lifts it into the Y-register. If you wish to clear a number from the X-register and replace it with another, use RDN instead of CLX.
COS	Cosine.
D-R	Degrees to radians.
DEC	Decimal. The range of the DEC function is extended to 12-digit octal integers between −777777777777 and +777777777777. In addition to fractional inputs, NaN and Inf will generate Data Error, regardless of trap settings.
DEG	Degrees mode.
DSE	Decrement and skip if less than or equal. If executed from the keyboard, DEC displays YES if the skip condition is *not* true, NO otherwise.
END	End of program.
ENG	Engineering mode. The acceptable range for ENG is extended to ENG 11.
ENTER^	ENTER^ is equivalent to RCL X. It does not disable stack lift. Its most common use on the HP-41 is to separate consecutive number entries—this is not necessary on the HP-71, as you can terminate number entry with [SPC] or [END LINE].
E^X	Natural exponential.
E^X-1	Natural exponential for arguments close to zero.
FACT	Factorial. The maximum input to the FACT function is 253 on the HP-71, compared to 69 on the HP-41.
FC?	Flag clear?
FC?C	Flag clear?—clear flag.
FIX	Fixed point mode. The acceptable range for FIX is extended to FIX 11.
FRC	Fractional part.
FS?	Flag set?
FS?C	Flag set?—clear flag.
GRAD	Grad mode. GRAD sets user flag 42, and clears flag 43, ▬ on the HP-41, but no GRAD annunciator is available. If you enter the BASIC environment after setting grad mode, the HP-71 will revert to degrees mode. Grad mode is cleared in favor of degrees mode or radians mode when you reenter the FORTH or HP-41 environments.

Table E-1. HP-41 Functions Included in the HP-41 Emulator (Continued)

GTO	Go to. GTO. and GTO.. ▪▪ not recognized by the emulator.
HMS	To hours-minutes-seconds.
HMS+	Hours-minutes-seconds plus.
HMS-	Hours-minutes seconds minus.
HR	To decimal hours.
INT	Integer part. Can also be spelled IP.
ISG	Increment and skip if greater.
LASTX	Recall from L register.
LBL	Label. LBL has no effect when executed from the keyboard. If you type, for example, LBL 22 [END LINE], the LBL is ignored, and 22 is entered into the X-register.
LN	Natural logarithm.
LN1+X	Natural logarithm for arguments close to 1.
LOG	Common logarithm.
MEAN	Means of summations. MEAN can produce the following errors: • Alpha Data, if any of the six statistics registers contains alpha data. • Invalid Arg, if N (the number of data entries) is zero. • Data Error, if any of the six statistic registers contain Inf or NaN.
MOD	y mod x.
OCT	To octal. The range of the OCT function is extended to integers in the range -68719476735 to +68719476735. Inf or Nan in the X-register generates Data Error, regardless of trap settings.
OFF	Turn off computer.
ON	Continuous on mode. ON sets flag -3 as well as user flag 44. The former sets HP-71 continuous on mode; the latter ensures that HP-41 programs can test the mode normally.
P-R	Polar-to-rectangular. Produces the IVL exception if y = Inf.
%	Percent.
%CH	Percent change.
PI	PI returns a 12-digit representation of pi.
POSA	Position in alpha.
PRA	Print alpha.
PRBUF	Print buffer.
PRFLAGS	Print flags and modes.
PRKEYS	Print keys file. PRKEYS executes the BASIC operation PLIST KEYS. If there is no keys file, fatal error ERR: File Not Found occurs.
PROMPT	Stop and display alpha.

Table E-1. HP-41 Functions Included in the HP-41 Emulator (Continued)

PRREG	Print registers.
PRREGX	Print registers by X. Data Error is |x|>999.
PRS	Print summation registers.
PRSTK	Print stack.
PRX	Print X.
PSE	Pause. Pressing any key during ■ pause halts program execution, which can be resumed with [RUN].
PSIZE	Programmable SIZE.
R^, RUP	Roll up. You can use either spelling for this function.
R-D	Radians to degrees.
R-P	Rectangular to polar. Produces the IVL exception if x and y are Inf.
RAD	Radians mode.
RCL	Recall register.
RCLFLAG	Recall flags.
RDN	Roll down.
REGMOVE	Register move.
REGSWAP	Register swap.
RND	Round. RND is extended to accomodate the STD format, and the FIX, SCI, and ENG formats up to 12 display digits. Attempting to round Inf or NaN produces the DATA ERROR message.
RTN	Return.
S+	Summation plus.
S-	Summation minus.
SCI	Scientific notation. The acceptable range for SCI is extended to SCI 11.
SDEV	Standard deviations. SDEV can produce the following errors: • Alpha Data, if any of the six statistics registers contains alpha data. • Invalid Arg., if N (the number of data entries) is 0 or 1. • Data Error, if any of the statictic registers contains Inf or NaN.
SF	Set flag.
SKPCHR	Skip characters.
SREG	Summation register set.
SREG?	Return address of first summation register.
SIGN	Sign.
SIN	Sine.
SIZE	Set size.
SIZE?	Return current size.
SQRT	Square root.

Table E-1. HP-41 Functions Included in the HP-41 Emulator (Continued)

ST+	Store plus
ST-	Store minus
ST*	Store multiply.
ST/	Store divide.
STO	Store into register.
STOFLAG	Store flags.
STOP	Stop program execution.
TAN	Tangent.
TIME	Time.
TONE	Tone.
VIEW	View register.
X^2	X-squared.
X=0?	
X#0? or X≠0?	Use either form.
X<0?	
X<=0?	
X>0?	
X>=0?	Not present in the HP-41.
X=Y?	
X#Y? or X≠Y?	Use either form.
X<Y?	
X<=Y?	
X>Y?	
X>=Y?	Not present in the HP-41.
X=NN?	
X#NN? or X≠NN?	Use either form.
X<NN?	
X<=NN?	
X>NN?	
X>=NN?	
X<>	X exchange.
X<>F	X exchange flags.

Table E-1. HP-41 Functions Included in the HP-41 Emulator (Continued)

X<>Y	X exchange Y.
XEQ	Execute.
XTOA	X to alpha. XTOA works the same way as on the HP-41, but some characters produced in the alpha register on the HP-71 may differ from those on the HP-41. Refer to the owner's manuals for the character codes for each calculator.
Y^X	Y to the X power.

Table E-2. Functions Unique to the Emulator

$	Alpha mode. Activate alpha mode.
A"	Load into the alpha register the characters following a space after A", up to but not including the next ". For example, A" HELLO" places the characters HELLO into the alpha register.
A,X	Alpha and X-register display mode. Execute A/X after each HP-41 command line is completed.
A/X	Alpha and X register display. Display the alpha register and the X-register in the format *alpha register* | *X-register*.
BASIC	Enter BASIC environment.
FORTH	Enter FORTH environment.
KEYS41	Merge the KEYS41 file with the current HP-71 keys file.
LOAD *file name*	Compile the program in the text file named *file name* into emulator memory. The [ATTN] key is disabled during execution of LOAD.
purge41	Purge HP-41 environment. Clear all HP-41 programs and data registers. Exit to FORTH environment.
PRINTER	Initialize printer flags. Search for an HP-IL printer (or PRINTER IS device) and set flags 55 and 21 if found, clear otherwise.
RUN	Begin program execution at the current program pointer position. Not programmable.
ST.	Stack display. Display the entire RPN stack, in the format: T Z Y X | L
XONLY	X-register display. Following each operation, display the X-register.

Note: The functions S-R, S+R, ENTER^R, and RCLXR are special versions of S-, S+, ENTER^, and RCLX. These special functions are used for translating HP-41 programs. Do not include these special functions in your HP-41 programs. When executed from the keyboard, the special functions perform the same operation as their standard counterparts.

The XBR function is another special translator function. It has no keyboard use. Executing XBR from the keyboard generates the message XBR: Compile Only.

Access to Other Environments

The HP-41 emulator allows you to use features of the BASIC and FORTH environments without leaving the HP-41 environment. This feature is derived from the fact that the HP-41 environment is actually a subset of the FORTH language system, which is designed to provide two-way interaction with the BASIC operating system.

The FORTH words BASICX and BASICF enable you to send commands to the BASIC interpreter, and to return numeric data from BASIC to the HP-41 floating-point stack. The syntax for these functions is:

```
" command string" BASICX
```

```
" numeric expression" BASICF
```

Notice the required space between the first quotes and the beginning of the *command string* or the *numeric expression*.

BASICX sends the command string to the BASIC interpreter, which executes the commands, then returns control to the HP-41 environment. For example, to determine the amount of HP-71 memory available, you can type

```
" MEM" BASICX [END LINE]
```

The display shows the current available memory, in bytes, for the duration of the current delay setting, and then returns to the X-register display.

Examples:

`" DELAY 0,0" BASICX`	Change the delay setting to 0,0.
`" COPY DATA TO :TAPE"BASICX`	Copy the file DATA to a cassette drive
`" CAT ALL" BASICX`	List current HP-71 files.

BASICF returns the value of the *numeric expression* to the X-register, lifting the stack.

Example: To enter the amount of available memory into the X-register, execute:

```
" MEM" BASICF
```

The BASIC keyword FORTHF is the reverse of BASICF—it reads the current value from the X-register into the BASIC environment. Using FORTHF and BASICF together enables you to use BASIC operations on HP-41 data.

Example: The following statement uses the BASIC CEIL function to compute the smallest integer greater than or equal to the value in the X-register, and then place that value into the X-register:

```
" CEIL(FORTHF)" BASICF
```

All of the HP-41 Pac's built-in FORTH dictionary words can be executed directly from the HP-41 environment. You should be aware, however, that the HP-41 emulator uses a vectored form of the word NUMBER; all numeric entries are sent to the floating-point stack instead of to the integer data stack, unless the number contains a double-length indicator (, or /). In addition the words 0, 1, 2, and 3 are redefined to floating point versions. In general, you should usually use the FORTH environment for FORTH operations. You can access HP-41 commands in FORTH by selecting the HP-41 vocabulary named HP-41V.

Appendix F

Guidelines for Running HP-41 Programs on the HP-71

This appendix summarizes how to use the HP-41 Translator Pac to run programs written for the HP-41 (for example, Users' Library programs that were written for the HP-41). The major topics covered are:

- Transferring your HP-41 program to the HP-71.
- Running the program. Instructions for running the program on the HP-41 may describe keystrokes that are done somewhat differently when running the program on the HP-71.

Transferring HP-41 Programs

There are three ways to enter an HP-41 program into the HP-71. You can:

- Directly transfer the program from HP-41 memory using HP-IL.
- Transfer the program from a mass storage medium (magnetic cards, tape, or flexible disc) to the HP-71.
- Type the program into the HP-71 using an HP-41 printed program listing.

The following instructions summarize the steps described in section 3 of this manual.

Directly Transferring an HP-41 Program

To directly transfer a program from HP-41 memory to the HP-71:

1. Connect both calculators together using their HP-IL modules.
2. Use the READ41 program to read the HP-41 program from HP-41 memory into the HP-71 as a text file. Follow steps 1 through 8 on pages 45 and 46.
3. When you've completed step #8 of the READ41 instructions, continue with steps 3 through 5 of the TRANS41 instructions on pages 42 and 43. Don't run the program, though, until you've read the rest of appendix F.

Transferring a Program From Mass Storage

The program to be transferred from mass storage must be in the form of a program-text file. A program-text file is an HP-71 text file containing an HP-41 program (for example, a file created by the text editor containing an HP-41 program listing). Users' Library HP-41 programs ordered specifically for the HP-71 are recorded on the mass storage medium in program-text file format. However, HP-41 programs recorded by the HP-41 on magnetic cards, tapes, or flexible discs cannot be used—those programs must be loaded into the HP-41 and transferred as described above.

To transfer an HP-41 program-text file from mass storage to the HP-71:

1a. **If the program is recorded on tape or a flexible disc**, connect the HP-71 to the mass storage device using HP-IL.

1b. **If the program is recorded on HP-71 magnetic cards**, copy it from the cards to HP-71 memory by typing:

COPY CARD TO *file name* [END LINE]

2. Follow steps 1 through 5 for running the TRANS41 program, on pages 41 through 43. If you copied your program from HP-71 magnetic cards to HP-71 memory, be sure to use the *file name* you used in step 1b, above. If you are using an HP-IL cassette drive or disc drive, make sure you use the appropriate file specifier in response to the TRANS41 prompt HP-41 Program File?.

3. Read the rest of Appendix F before running the program.

Entering a Program From the Keyboard

1. Run the HP-71 text editor, using steps 1 through 4 on page 39 to prepare to type in the program.

2. Type the HP-41 program by copying the program listing line-by- line, without line numbers. (Refer to pages 40 and 41, if necessary, for additional information.)

3. Follow steps 1 through 5 for running the TRANS41 program, on pages 41 through 43.

4. Read the rest of Appendix F before running the program.

HP-41 Program Instructions and Keystrokes

HP-41 program instructions, such as instructions included with HP Users' Library programs, contain:

- A program description.
- User instructions—step-by-step keystrokes for program execution.
- Examples and results.
- Documentation of the program's use of data registers, status messages, and flags.
- A list of HP-41 key assignments made by the program.

Program Description

The program description will be the same for the two calculators.

User Instructions

To execute an HP-41 program on the HP-71, you must perform the same step-by-step operations as described in the program documentation. However, the actual keystrokes may be somewhat different for the HP-71 due to the different keyboards of the two calculators. In addition, different program authors use different conventions in their documentation to describe HP-41 keystrokes. So, the guidelines described here may need to be adapted to the instructions for your program.

Use the following general rules to convert HP-41 keystroke instructions:

Initializing the Size. Most HP-41 programs require you to set the number of data registers to a minimum SIZE. On the HP-71, you type:

SIZE *nnn* [END LINE]

where *nnn* is the number (1,2, or 3 digits) specified by the program instructions.

Starting the Program. HP-41 programs are referenced by *global* and *local* labels within the programs. Global labels, consisting of alphanumeric strings, can be called at any time using the GTO or XEQ functions. Local labels, consisting of numbers 00 through 99, uppercase letters A through J, or lowercase letters a through e, can only be called when the HP-41 program pointer is positioned within the program containing the label.

The GTO function positions the program pointer at a program label. You can then begin program execution at that position by pressing [RUN] or by typing RUN [END LINE]. The XEQ function is equivalent to GTO followed by [RUN]; it moves the program pointer to the specified label and begins execution at that position.

For a global label, the conversion from HP-41 keystrokes to HP-71 keystrokes is:

HP-41 Keystrokes	HP-71 Keystrokes
[GTO] [ALPHA] *label* [ALPHA]	GTO"*label*" [END LINE]
[XEQ] [ALPHA] *label* [ALPHA]	XEQ"*label*" [END LINE]
[R/S]	[RUN] or RUN [END LINE]

For the HP-71 keystrokes, quotation marks substitute for the [ALPHA] key. There must not be a space between the GTO or XEQ and the quoted string naming the label.

Examples:

[GTO] [ALPHA] STD [ALPHA] becomes GTO"STD" [END LINE]

[XEQ] [ALPHA] STD [ALPHA] becomes XEQ"STD" [END LINE]

For local labels, use the following conversion:

HP-41 Keystrokes	HP-71 Keystrokes
[GTO] *number*	GTO *number* [END LINE]
[XEQ] *number*	XEQ *number* [END LINE]
[GTO] [ALPHA] *letter* [ALPHA]	GTO *letter* [END LINE]
[XEQ] [ALPHA] *letter* [ALPHA]	XEQ *letter* [END LINE]

Examples:

[GTO] 56 becomes GTO 56 [END LINE]

[XEQ] [ALPHA] A [ALPHA] becomes XEQ A [END LINE]

For additional information about program labels, refer to page 34.

Number Entry. When instructions involve entering a single number, type the number in HP-71 format (for example, 1234.5, -49, 56.2E-12). Terminate number entry by pressing [SPC] or [END LINE]. If the HP-41 program requires you to enter two or more numbers separated by [ENTER], on the HP-71 you can type the numbers together on the same line, separated by spaces. **Do not use** ENTER^ **to separate numbers on the HP-71**; using ENTER^ duplicates the entry and can cause the program to not work properly.

Examples:

1.234 [R/S] becomes 1.234 [RUN]

1.2 [CHS] 34 [EEX] 27 [R/S] becomes -1.234E27 [RUN]

4.567 [ENTER↑] 7.89 [R/S] becomes 4.567 7.89 [RUN]

89 [ENTER↑] 97 [ENTER↑] 101 [R/S] becomes 89 97 101 [RUN]

65 [ENTER↑] [R/S] becomes 65 ENTER^ [RUN]

In the last example, above, [ENTER↑] is used to duplicate the entry 65, rather than to separate numbers; on the HP-71, the emulator ENTER^ function is used.

For additional information about number entry and used of ENTER^, refer to pages 24 through 27.

Alpha Entry. The only difference between alpha mode on the HP-41 and the HP-71 is the way alpha mode is entered and exited.

Operation	HP-41 Keystrokes	HP-71 Keystrokes
Entering alpha mode	[ALPHA]	three methods: $ [END LINE] AON [END LINE] [f] [END LINE] (in USER mode with KEYS41 the active keys file)
Exiting alpha mode	[ALPHA]	[END LINE]

Refer to table 2-6 on page 32 for a list of keys active in alpha mode.

Executing Functions. Some HP-41 programs require you to execute functions. In general, to execute a function on the HP-71, type in the function name (with any parameters) as it appears in the program instructions or program listing.

Examples:

[SIN]	becomes	SIN [END LINE]
[STO] 99	becomes	STO 99 [END LINE]
FS?C 21	becomes	FS?C 21 [END LINE]
SCI 9	becomes	SCI 9 [END LINE]

Refer to pages 23 through 28 for additional information on HP-41 emulator functions.

Program Examples and Results

HP-41 programs transferred to the HP-71 usually produce the same numerical and display results on both calculators. You should normally be able to follow step-by-step examples included with the HP-41 program instructions, and see results as described. However, there are some features of the HP-71 that can produce different results. Here is a brief summary (refer to Appendix E for a more complete discussion):

- Numerical results are computed with more accuracy on the HP-71 than on the HP-41. This can cause some results to differ in the last decimal place. For example, evaluating the expression

$$(\sqrt{2})^2 - 2$$

produces different results on the HP-41 and HP-71:

Calculator	Keystrokes	Results
HP-41	2 [ENTER↑] [√x] [■] [x²] 2 [−]	−1.00 −09
HP-71	2 SQRT X^2 2 -	-1.0000E-11

One answer is 100 times the other, yet the results are effectively the same.

- Alpha displays may differ because of the absence of digit separators and ■ radix choice on the HP-71. Also, certain special HP-41 characters appear as different characters on the HP-71.
- The HP-71 handles errors with more sophistication than the HP-41. This can result in different error messages, and in some cases, different results, depending on the settings of the HP-71 IEEE mathematical exception traps.

Examples of cases where programs run correctly on the HP-71, and yet do not produce the same example results, occur frequently in games programs that use random numbers to control the "play" of the game. Such routines typically are very sensitive to the number of digits used by the calculator in its computation. While the HP-41 game will play correctly on the HP-71, sample plays will very likely not follow the program examples exactly.

Data Registers, Status Messages, and Flags

Use of data registers, status messages, and flags are the same on both calculators.

Converting HP-41 Key Assignments

HP-41 programs can have two kinds of "built-in" key assignments:

- **Global key assignments.** If the HP-41 is in USER mode when a program is loaded from magnetic cards or mass storage, global label assignments stored within the program are automatically activated.
- **Automatic assignment of local alpha labels.** Pressing a key in either of the two top rows on the HP-41, or a shifted key in the top row, automatically executes the local alpha label corresponding to the key's alpha mode character.

Neither of these types of automatic assignments can be transferred to the HP-71. If the instructions for an HP-41 program assume that these key assignments are active, you must translate the instruction keystrokes into the appropriate XEQ function. For example, an HP-41 program instruction might tell you to press the [1/x] key, which on the HP-41 would cause execution at the local label B. On the HP-71, you must type XEQ B [END LINE].

Use the following table to interpret instructions using local label assignments:

HP-41 Key	Local Label	HP-71 Keystrokes
[Σ+]	A	XEQ A [END LINE]
[1/x]	B	XEQ B [END LINE]
[√x]	C	XEQ C [END LINE]
[LOG]	D	XEQ D [END LINE]
[LN]	E	XEQ E [END LINE]
[X<>Y]	F	XEQ F [END LINE]
[R↓]	G	XEQ G [END LINE]
[SIN]	H	XEQ H [END LINE]
[COS]	I	XEQ I [END LINE]
[TAN]	J	XEQ J [END LINE]
[■][Σ-]	a	XEQ a [END LINE]
[■][Y^x]	b	XEQ b [END LINE]
[■][x^2]	c	XEQ c [END LINE]
[■][10^x]	d	XEQ d [END LINE]
[■][e^x]	e	XEQ e [END LINE]

For global label assignments, no standard conversion is possible, since the programmer can assign any label to any key. The program instructions should include a list of the global label assignments, and you must substitute the approriate XEQ function for each assignment. For example, if the program expects LBL"PROG" to be assigned to the [TAN] key, you must type XEQ"PROG" [END LINE] when the instructions call for pressing [TAN].

You can make HP-71 keys assignments that mimic any of the HP-41 key assignments (refer to pages 28 and 29). For example, if you wish the HP-71 [E] key to correspond to the HP-41 [E] key (actually, the [LN] key), type:

```
KEY"E"," XEQ E" END LINE
```
(in the BASIC environment)

or:

```
" KEY'E',' XEQ E'" BASICX END LINE
```
(in the HP-41 environment)

To assign the [=] key to execute the program PROG, type:

```
KEY'=',' XEQ"PROG"' END LINE
```
(in the BASIC environment)

Because of the special use of quotes by the HP-41 system, this assignment can be made conveniently only from the BASIC environment.

Subject Index

Page numbers in **bold** type indicate primary references.

BASIC Keywords by Category

This list shows all BASIC keywords by functional category. All BASIC keywords and their defininitions appear in appendix C, in alphabetic order.

Keyword	Description
BASIC to FORTH	
FORTH	Transfers HP-71 operation to the FORTH environment.
FORTH$	Returns to ▪ BASIC string variable the contents of a string in the FORTH enivronment.
FORTHF	Returns to ▪ BASIC numeric variable the contents of the FORTH floating-point X-register.
FORTHI	Returns to ▪ BASIC numeric variable the value on the top of the FORTH data stack.
FORTHX	Executes ▪ FORTH command string.
Editor	
DELETE#	Deletes one record from a text file.
EDTEXT	Invokes the text editor.
FILESZR	Returns the number of records in a text file.
INSERT#	Inserts one record into ▪ text file.
MSG$	Returns the message string corresponding to ▪ specified error number.
REPLACE#	Replaces one record in ▪ text file.
SCROLL	Scrolls the display and waits for ▪ key to be pressed.
SEARCH	Finds a string in ▪ text file.
BASIC/FORTH to HP-41	
HP41	Transfers HP-71 operation to the HP-41 environment.

FORTH Words by Category

This list shows all FORTH words by functional category. Some words appear in more than one category. All FORTH words and their defininitions appear in appendix D, sorted by name in ASCII order.

General

Dictionary Management

ALLOT
CONTEXT
CURRENT
DEFINITIONS
DSIZE
FENCE
FORGET
FORTH
GROW
HERE
NALLOT
PAD
ROOM
SHRINK
VOCABULARY
XSIZE

System

>BODY
?STACK
ABORT
ABORT"
ASSEMBLE
BYE
CHIRP
CLOCK
DECIMAL
DEG
DEGREES
DEPTH
ENG
EXECUTE
FIND
GRAD
HEX
LATEST
QUIT
RAD
RADIANS
SCI
TIB
TOGGLE
TRAVERSE

Control Structures

BEGIN . . . UNTIL
BEGIN . . . WHILE
  . . . REPEAT
CASE . . . OF. . . ENDOF
  . . . ENDCASE
DO . . . +LOOP
DO . . . LOOP
IF . . . THEN
IF . . . THEN
  . . . ELSE
LEAVE

Memory

!
+!
4N@
?
@
C!
C@
C@+
CMOVE
CMOVE>
FILL
N!
N@
NFILL
NMOVE
NMOVE>
RCL
S!
SMOVE
STO

Interpretation

'
(
INTERPRET

BASIC System Access

BASIC
BASIC$
BASICF
BASICI
BASICX
BYE
CLKEYS

HP-41 Access

HP41
HP41V

Return Stack

>R
I
J
R>
R@
RP!
RP0
RP@

Defining Words

:
;
CONSTANT
CREATE
EXIT
FCONSTANT
FVARIABLE
STRING
STRING-ARRAY
VARIABLE

Compilation

,
?COMP
C,
COMPILE
DLITERAL
DOES>
FLITERAL
IMMEDIATE
LITERAL
NOP
SMUDGE
STATE
[
[']
[COMPILE]
]

Files

File Manipulations

+BUF
ADJUSTF
BLOCK
CLOSEALL
CLOSEF
CREATEF
EOF
FINDF
FLUSH
LOADF
OPENF
SYNTAXF

General Purpose Buffers

CONBF
EXPBF
FINDBF
KILLBF
MAKEBF

Arithmetic

Single Length

```
*
*/
*/MOD
+
-
/
/MOD
1+
1-
2*
2+
2-
2/
5+
5-
ABS
FTOI
MOD
NEGATE
```

Double Length

```
D+
D-
DABS
DNEGATE
S->D
```

Mixed Length

```
M*
M/
M/MOD
UM*
UM/MOD
```

Floating Point

```
%          HMS
%CH        HMS+
1/X        HMS-
10^X       HR
ACOS       IP
ASIN       ITOF
ATAN       LGT
CHS        LN
COS        LN1+X
D-R        LOG
DEC        MOD
E^X        OCT
E^X-1      PI
F*         P-R
F+         R-D
F-         R-P
F/         SIGN
FABS       SIN
FACT       SQRT
FP         TAN
           X^2
           Y^X
```

Logical

```
AND
NOT
OR
XOR
```

Stack Manipulations

Single Length

```
DEPTH
DROP
DUP
OVER
PICK
ROLL
ROT
S0
SP!
SP0
SP@
SWAP
```

Double Length

```
2DROP
2DUP
2OVER
2SWAP
```

Floating Point

```
CLST
CLX
FDROP
FENTER
LASTX
RDN
RUP
X<>Y
```

Comparisons

Single Length

```
0<         >
0=         ?DUP
0>         MAX
<          MIN
<>         U<
=
```

Double Length

```
D<
```

Floating Point

```
X#0?       X<Y?
X≠0?       X=0?
X#Y?       X=Y?
X≠Y?       X>=0?
X<=0?      X>=Y?
X<=Y?      X>0?
X<0?       X>Y?
```

String

```
S<         S=
```

Input/Output

Constants

```
0
1
2
3
BL
```

Numeric-Input Conversion

```
BASE
CONVERT
DIGIT
NUMBER
```

Numeric Output

```
.
.S
BASE
D.
D.R
F.
H.
U.
```

Number Formatting

```
#
#>
#S
<#
ENG
FIX
HOLD
SCI
SIGN
STD
```

Character Input

```
?TERMINAL
COUNT
ENCLOSE
EXPECT96
KEY
QUERY
WORD
'STREAM
```

Character Output

```
-TRAILING
."
.<
CR
EMIT
SPACE
SPACES
TYPE
```

HP-IL

```
ENTER
OUTPUT
PRIMARY
SECONDARY
```

User Variables

```
#TIB
>IN
BASE
BLK
CONTEXT
CURRENT
FIRST
L
LIMIT
LINE#
OKFLG
ONERR
PREV
PRIMARY
SCRFIB
SECONDARY
SPAN
STATE
T
USE
WARN
WIDTH
X
Y
Z
```

String Words

```
"
ASC
CHR$
CRLF
END$
FSTR$
LEFT$
MAXLEN
NULL$
POS
RIGHT$
S!
S<
S<&
S=
S>&
STR$
SUB$
VAL
```

HEWLETT PACKARD

Portable Computer Division
1000 N.E. Circle Blvd., Corvallis, OR 97330, U.S.A.

European Headquarters
150, Route Du Nant-D'Avril
P.O. Box, CH-1217 Meyrin 2
Geneva-Switzerland

HP-United Kingdom
(Pinewood)
GB-Nine Mile Ride, Wokingham
Berkshire RG11 3LL

82490-90001 English

Printed in U.S.A. 1/85
82490-90002


XEQ" "
GTO" "
SF
CF
Σ+
Σ−
CLΣ
MEAN
SDEV
SQRT
XEQ
RTN
GTO
FS?
FC?
BEEP
x≷y
SIN
COS
TAN
10^x
CAT
1/X
e^x
LN
AON
ASIN
ACOS
ATAN
LOG
BACK
APPEND
RCL
STO
USER
VIEW
"_"
LAST X
ATTN
CLA
ASHF
RUP
RDN
HP-41